UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

SISTEMA EXPERTO PARA LA GENERACIÓN DE ACOMPAÑAMIENTOS MUSICALES

PROYECTO FIN DE CARRERA
INGENIERÍA SUPERIOR DE TELECOMUNICACIÓN

Autor: Pablo Martín Oñate
Tutor: Julio Villena Román

Julio 2009

Título: Sistema experto para la generación de acompañamientos musicales.

Autor: Pablo Martín Oñate

Tutor: Julio Villena Román

EL TRIBUNAL

Presidente:
José Jesús García Rueda

Secretario:
Iria Estévez Ayres

Vocal:
Óscar Quevedo Teruel

Realizado el acto de defensa del Proyecto Fin de Carrera el día 21 de Enero de 2008 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Vocal Fdo: Secretario

Índice

1. INTRODUCCIÓN

1.1 Motivación y objetivos

En este proyecto se va a realizar un sistema de composición musical computacional, mediante el uso de la inteligencia artificial. La herramienta por la que se ha optado para llevar a cabo la implementación de dicho compositor ha sido un sistema experto.

La funcionalidad del sistema consiste en generar una línea musical de acompañamiento a partir de otra línea melódica dada, de tal forma que ambas se puedan combinar y coincidan en un equilibrio armónico. Se obtendrá, por tanto, una pieza musical compuesta por dos voces, por un lado la voz principal generada externamente, y por otro la segunda voz generada por el compositor a partir de la principal.

Para decidir los acordes que constituirán el acompañamiento de la melodía principal, se utilizarán unas reglas armónicas preestablecidas, creadas a partir de normas clásicas de la armonía musical.

Este sistema tiene dos objetivos fundamentales. El primero consiste en ofrecer una plataforma musical, que permita poder decidir el tipo de composición que se quiere realizar, sin necesidad de modificar la base del conocimiento que constituye el sistema experto.

El segundo objetivo es la implementación de una regla armónica que forme parte de la base del conocimiento, y que junto con una estructura modular auxiliar, permita darle una funcionalidad musical inicial al compositor.

El sistema modular auxiliar será una estructura compuesta por una serie de bloques que se comunicarán e interactuarán entre sí, y cuya finalidad será

preparar toda la información necesaria para que la regla pueda ejecutarse correctamente, de manera que se obtenga la melodía de salida esperada.

Para la realización del sistema compositor se necesitará hacer uso de dos bibliotecas Java específicas (JMusic y Jess), que permitirán desarrollar el código de manera que tenga funcionalidad y conocimiento musicales, y capacidad de decisión.

La funcionalidad musical del programa permitirá tanto la utilización de elementos básicos relativos a la música (notas, duración, etc.), como la manipulación y análisis de ficheros de audio en formato .MIDI. De esta manera se podrán abrir archivos MIDI, extraer sus notas, generar acordes, darles la longitud adecuada, etc. Todas estas actividades se realizarán mediante el uso de la biblioteca Java mencionada anteriormente, JMusic.

El conocimiento musical y capacidad de decisión necesarios a la hora de generar el acompañamiento de una línea melódica, se conseguirán gracias al lenguaje de inteligencia artificial Jess. Con esta herramienta, se podrá definir un conjunto de reglas capaces de aplicar un conocimiento de composición musical real, para decidir qué combinación de notas es la adecuada a la melodía de entrada.

En resumen, se puede concluir que mediante este sistema se pretende construir la estructura básica de un compositor musical, que funcione como un sistema experto y por tanto tenga un conjunto de reglas definidas de las cuales se seleccione, sin modificar dicha estructura, aquella que sea la adecuada para realizar la tarea musical que se desee. Adicionalmente debe estar constituido por una regla, al menos, para poder comprobar su funcionamiento como compositor.

1.2 Estructura del documento

La estructura del proyecto es la siguiente:

El presente capítulo 1, que constituye la introducción, tiene como finalidad exponer de forma breve los objetivos que se propone alcanzar este sistema, así como mostrar la estructura del mismo.

En el capítulo número 2, titulado *Estado del arte*, se hace una introducción al concepto de inteligencia artificial así como algunas de sus teorías, centrándose sobre todo en la rama de ésta que aquí interesa, los sistemas expertos.

Se explicarán en detalle cuáles son los fundamentos básicos de los sistemas expertos, sus partes, las pautas de trabajo y las ventajas y limitaciones de los mismos. También se presentarán ejemplos de proyectos que se han basado en los sistemas expertos.

En otro punto del capítulo se hablará sobre la música, de manera independiente a la tecnología. Se desarrollará la relación existente entre la música y la actividad del cerebro. También se tratará la composición musical y la armonía a lo largo de la historia.

Todo esto permitirá establecer un contexto para poder situarse y comprender mejor el concepto que se quiere exponer en última instancia.

En el último punto, se relaciona finalmente la música con la inteligencia artificial. Se separará por un lado la relación existente entre la inteligencia artificial y el conocimiento, exponiendo las teorías para plasmar dicho conocimiento en un ordenador, y por otro lado su relación con la composición musical.

En el tercer capítulo, *Diseño del sistema*, se presentará en primer lugar cuáles han sido los problemas a los que nos hemos enfrentado a la hora de plantear el proyecto y en el momento de empezar a desarrollarlo.

También se expone cuál es la estructura del sistema y se explican en detalle los diferentes bloques que lo componen y cómo interactúan entre sí para darle la funcionalidad al compositor.

El cuarto capítulo es la *Implementación del sistema*. En él se presentan las dos herramientas fundamentales empleadas en la construcción del proyecto, JMusic y Jess. También se explicarán en detalle todas las clases, y métodos que constituyen el compositor.

Por último se expondrán las conclusiones obtenidas a lo largo de la realización del proyecto, así como posibles trabajos futuros y posibles módulos a añadir para completar la funcionalidad del compositor.

2. ESTADO DEL ARTE

2.1 Música

2.1.1 La música y el cerebro

La música ha sido compañera del ser humano desde los comienzos de su historia. Existen hipótesis de que este comienzo tiene que ver con la imitación de los sonidos de la naturaleza, como el canto de las aves y también de la naturaleza interna del ser humano, como por ejemplo el ritmo natural de los latidos del corazón; las últimas teorías concernientes a los comienzos del arte le dan mucha importancia a este último punto (formas percibidas internamente), refiriéndose a estas influencias como “entópicas”.

Las teorías de los antiguos filósofos griegos concuerdan con las especulaciones de los eruditos en la época medieval, definiendo a la música como un conjunto de tonos ordenados de manera horizontal (melodías) y vertical (armonía).

El concepto de armonía (varias notas a la vez) se encontraba presente ya en la música prehistórica. A lo largo de ésta la música formaba parte de celebraciones y rituales, lo cual no difiere mucho de los usos que se le da hoy en día. De hecho nuestro sistema nervioso es prácticamente el mismo, por lo que aquello que emociona al ser humano es de naturaleza universal. Debido a esto se puede afirmar que la música está relacionada normalmente con la expresión de emociones, algo puramente humano, jugando su intelecto un importante papel en actividades musicales. Se podría considerar que las emociones son el significado de la música.

En nuestro campo de interés se puede definir la música como una actividad intelectual, como una habilidad esencial de la mente humana que requiere sofisticados mecanismos de memoria, involucrando tanto

manipulaciones conscientes de conceptos, como accesos subconscientes a millones de nodos neurológicos pertenecientes a una compleja red. Así, se considerará que las reacciones emocionales a la música vendrán de algún tipo de actividad intelectual. La música estimula los elementos cognitivos del lenguaje, tonalidad, emoción y ritmo del cerebro.

Como se ha dicho la música es una actividad intelectual, y supone una estimulación de ciertas áreas del cerebro, en base a lo que se oiga, tipo de música, sonido… En relación a estas ideas, se han realizado estudios en los que se demuestra que la música influye de manera significativa en la actividad cerebral así como en el desarrollo de ciertas capacidades, como por ejemplo matemáticas, la memoria, memoria verbal, literatura, visión espacial e incluso la propia inteligencia (TARKO, 2006). Esta influencia es mayor cuanto más joven es el cerebro.

La música no es percibida exclusivamente a través de los oídos, sino también por ejemplo a través de nuestra piel. Se ha descubierto un aumento del flujo de sangre y oxígeno en distintas partes del cerebro cuando se produce música o cuando se escucha. Esto implica que existe actividad en distintas áreas del cerebro cuando se percibe o se hace música.

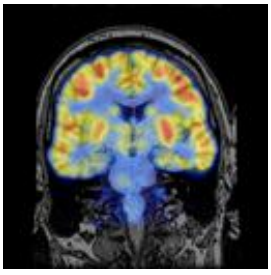
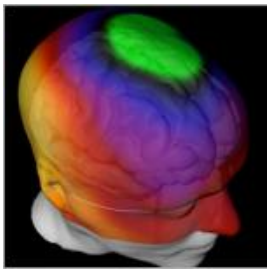
La actividad cerebral en respuesta a los estímulos puede medirse mediante la actividad de las neuronas (MIRANDA, 2003). Aquí se van a presentar tres métodos comunes de medida:

- **PET (Positron Emision Tomography)**: mide la actividad cerebral escaneando el flujo de un material radiactivo previamente insertado en el flujo sanguíneo del individuo observado. A pesar de su eficiencia no es un método muy popular por el desconocimiento de los posibles efectos secundarios que dicho material pueda tener en la salud. Proporciona una sección clara del área del cerebro donde el flujo sanguíneo es más intenso durante el proceso auditivo y

permite verificar que escuchar música e imaginar escuchar música activan diferentes partes del cerebro.

- **ERP (Event Related Potential)**: utiliza pequeños electrodos en contacto con el cráneo para medir la actividad eléctrica del cerebro. Es más seguro que PET. Da un nivel de voltaje en el tiempo de la actividad eléctrica de las áreas del cerebro donde los electrodos han sido colocados. Permite comprobar que el cerebro espera ciertas secuencias de estímulos, por lo que si escuchamos “el músico compuso la canción”, el cerebro tenderá a funcionar de manera estable, mientras que si escuchamos “el perro compuso la canción” el cerebro mostrará respuestas eléctricas negativas significativas.

- **Magnetoencefalografía (MEG):** esta técnica de escaneo del cerebro mide los pequeños campos electromagnéticos que fluyen desde nuestra cabeza. Las corrientes iónicas de una única neurona son demasiado pequeñas como para generar campos magnéticos medibles. Sin embargo, cuando las neuronas actúan en grupo y lanzan de forma sincronizada ráfagas de iones las unas a las otras, producen pequeños campos que sí pueden ser detectados. El número de neuronas necesario para generar una señal magnética detectable es de unas 50.000. La principal ventaja de este método es la velocidad, ofrece la posibilidad de medir la actividad del cerebro prácticamente en tiempo real, milisegundo a milisegundo. Esto permite a los científicos observar cómo trabaja el cerebro a medida que la información llega desde los oídos a las distintas partes del cerebro.


PET *ERP* *MEG*

Figura 1: Técnicas de medición de actividad cerebral

2.1.2 Composición musical

En primer lugar se va a exponer de manera breve la historia de la música occidental y la composición musical. Esto permitirá entender de dónde proceden las reglas armónicas que se han utilizado en este trabajo, puesto que no en todas las etapas de la historia musical se han desarrollado las mismas técnicas de composición y armonía, siendo, por otro lado, fundamentales cada una de estas etapas para la evolución hacia la siguiente.

En el mundo occidental (Mailxmail.com, 2002) la música tiene sus raíces en la Grecia antigua donde la música aparece como un fenómeno ligado a la necesidad del hombre de comunicar sentimientos y vivencias. La música coral era el elemento básico en la educación de los jóvenes espartanos y era un elemento fundamental dentro de las tragedias griegas. Era una época en la que predominan los elementos rítmicos sobre los melódicos y la voz humana tenía una clara primacía sobre los instrumentos, que eran pocos y no demasiado variados. Sin embargo, se conocían ya instrumentos de viento, como la flauta de Pan, y de cuerda como las cítaras o arpas.

Figura 2: Representación de danza tradicional en la Grecia Clásica

Fueron los griegos los primeros en imaginar, en el siglo VI a.C., un sistema de notación relativamente conciso, que utilizaba como signos los caracteres de un alfabeto arcaico en diferente posición, según respondieran al sonido natural, a un semitono o a la elevación de un cuarto de tono.

Años más tarde, durante los siglos V al XV d.C., se desarrolló lo que se denomina la música medieval. Durante este periodo nacieron dos tendencias diferentes, la cristiana y la profana.

La música cristiana surgió debido a la profunda preocupación por la religión, que favoreció el desarrollo de la música litúrgica a partir del siglo XI y cuya máxima expresión es el canto gregoriano, una melodía, cantada al unísono, que traducía el sentimiento religioso por la propia fuerza de su elocuencia, sin apoyos armónicos ni rítmicos. La iglesia cristiana primitiva, consideraba que las formas y tipos de música vinculados a celebraciones, festivales, espectáculos, etc., eran inapropiadas para la iglesia y no tanto por el disgusto que pudiese ocasionarles la música propiamente dicha como por la necesidad de apartar al creciente número de conversos de todo cuanto estuviese asociado con su pasado pagano (GROUT y PALISCA, 1995). En este tiempo también se crearon los primeros sistemas de notación musical, es decir, los neumas. Los neumas son una combinación de barras y puntos que aparece en ciertos fragmentos de los manuscritos a mediados del siglo XI y que deriva de los acentos de la escritura literaria. Con el tiempo estos sistemas se fueron perfeccionando y homogeneizando hasta derivar en el pentagrama y en las siete notas clásicas.

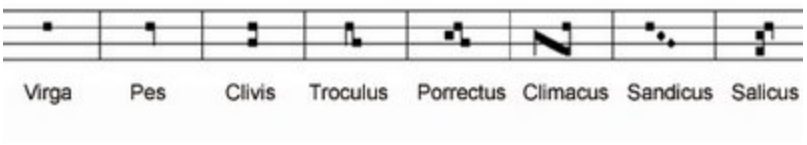


Figura 3: Neumas simples

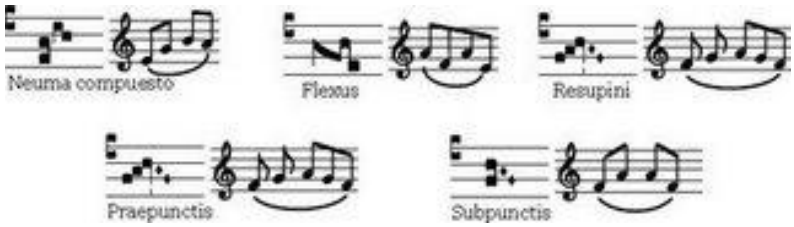


Figura 4: Neumas compuestos

A su vez, la música profana o trovadoresca se estaba desarrollando, sin embargo, no se poseen muestras de muchas canciones profanas de la Edad Media anteriores al siglo XI, aunque el repertorio debió de ser muy abundante, a juzgar por la cantidad de edictos eclesiásticos que lo condenaban.

A partir del siglo XII y a medida que se afirmaban y definían los modos y la música adquiría un carácter tonal, surge la polifonía, es decir, el arte de combinar sonidos distintos. Sobre una voz que sostiene la melodía se van agregando otras voces, en variaciones horizontales y verticales.

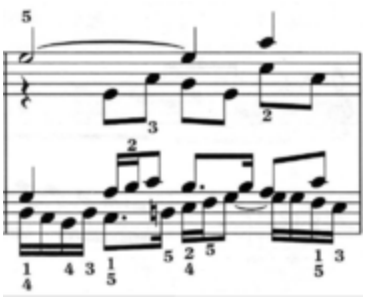


Figura 5: Ejemplo de melodía con varias voces

La siguiente etapa en la historia de la música es el Renacimiento, que es el movimiento cultural que surge en Europa en el siglo XIV, y que muestra como característica esencial su admiración por la antigüedad grecolatina. En el plano de la música, este movimiento aportó algunos cambios. La música profana adquirió más importancia y se desarrolló con gran fuerza la forma operística, se crearon nuevos instrumentos como el clavicordio y el clavicémbalo. También se refuerzan los aspectos rítmicos de la música, así como se libera y se perfecciona la armonía.

Durante los siglos XVII y XVIII se desarrolla el periodo Barroco. En esta época la música es utilizada por las monarquías absolutistas para

engrandecerse frente al pueblo. En este periodo se definieron nuevas formas musicales que serían básicas en la evolución posterior de la música como la sonata, la sinfonía o el concierto con solista. Surgió aquí la figura del castrati, que conservaban el timbre agudo y la tesitura elevada de un muchacho con la potencia respiratoria de los adultos. Los instrumentos más característicos del Barroco son la familia de cuerda de los violines, que sufrieron una gran transformación.

En esta etapa cabe destacar al compositor Johann Sebastián Bach, que es el pilar sobre el que se cimienta toda la música posterior. Más adelante se hablará de su trabajo y de la influencia que ha ejercido sobre las reglas de composición que existen hoy.

Figura 6: Johann Sebastian Bach

Tras el Barroco llegó un breve periodo que abarcó la segunda mitad del siglo XVIII, el clasicismo. Tuvo su principal expresión en Alemania, con autores como Mozart, Gluck o Haydn. El periodo clásico aportó mejoras técnicas en todos sus aspectos y la adopción de un ideal de perfección basado en el equilibrio entre el sentimiento y la razón.

Figura 7: Mozart

La siguiente etapa musical fue el Romanticismo, que ocupó prácticamente todo el siglo XIX. Se caracterizó por la primacía de los elementos subjetivos sobre la razón, el ideal del artista romántico es expresar su mundo interior directa y sinceramente. Se basaron con frecuencia en melodías y formas musicales de raíz popular. La música romántica tendía a ser programática, es decir, pretendía narrar directa o indirectamente una historia. A ello se debió el auge de la ópera y la creación de los poemas sinfónicos. Un hecho importante durante el periodo romántico fue la extinción de la figura del compositor adscrito al servicio de una iglesia o de un príncipe, algo que se mantuvo inmutable hasta Haydn y que incluso el mismo Mozart no logró romper sin graves consecuencias.

Con Beethoven, el compositor empezó a ser un artista libre que componía por propia decisión y que asumía una responsabilidad como creador independiente ante su sociedad. Aceptaba el mecenazgo pero no la servidumbre y, en general, a lo largo de todo el siglo el compositor intentaba ganarse la vida de manera liberal a través de los ingresos que sus obras pudieran proporcionarle. También es de destacar la primacía del piano como instrumento por excelencia y la ampliación de la orquesta en busca de nuevas riquezas sonoras.

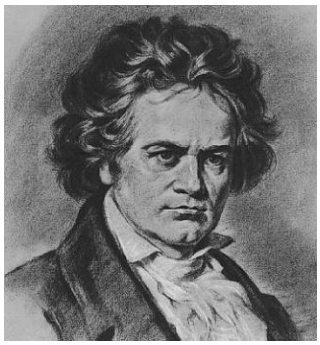


Figura 8: Ludwig van Beethoven

Para finalizar con la historia de la música, se va a hablar del siglo XX. En este siglo se puede hablar de un auténtico "estallido" de la música. Los principios más sólidos son los desconocidos, las ideas más extravagantes son intentadas y las escuelas más diversas conviven sin problemas. La música culta o elaborada se intelectualiza y se hace elitista. Al mismo tiempo que los grandes medios de difusión alcanzan a todos los públicos, los compositores se alejan de éstos y se refugian en ámbitos creativos cerrados. Se encuentran en el siglo XX los últimos brotes del romanticismo como Mahler y Richard Strauss. La preocupación por reflejar la realidad provoca el movimiento verista en la ópera italiana que produce un único compositor, Puccini. La llamada escuela de Viena rompe con toda la tradición anterior y crea el dodecafonismo, música en doce sonidos básicos, y la música atonal. Este camino, es seguido luego por casi todos los compositores contemporáneos, como por ejemplo Stravinski.

2.1.3 Armonía

A continuación se va a explicar lo que significa armonía y a desarrollar brevemente su historia.

La palabra armonía tiene diversas definiciones. Los griegos empleaban este término para representar el “perfecto equilibrio” en el ser humano tanto en lo referente a su estado físico como intelectual (RASO DEL MOLINO, 2000), lo que producía una determinada forma de actuación durante el transcurso de su vida. Más cerca en el tiempo, Luis de Góngora dijo “aún, ante una infame turba de nocturnas aves gimiendo tristes sonidos y volando grave, existe la armonía”,

y Leibniz, considera que “el universo tiene la suficiente armonía, para que todas las especies y elementos que lo forman, puedan existir y coexistir tanto individual como colectivamente con o sin influencias entre ellos”. Por tanto la idea de armonía que se puede deducir de unas y otras definiciones viene resumida en la siguiente definición: armonía es “la conveniente proporción y correspondencia entre unas cosas y otras”.

En el mundo de la música, la armonía no deja de significar lo mismo que se ha expuesto anteriormente, es un equilibrio perfecto existente a la hora de mezclar sonidos.

Al hablar de armonía hay que mencionar también la idea de contrapunto, puesto que ambos conceptos están íntimamente relacionados. El contrapunto es la habilidad de poder conducir varias líneas melódicas simultáneas de acuerdo a la estructura y sistema armónico empleado en una composición.

El estudio de la armonía hace referencia, a menudo, a las progresiones armónicas, que son el movimiento melódico de un conjunto de tonos, tocados simultáneamente, a otro, y a los principios estructurales que rigen tales progresiones. Tradicionalmente, la armonía funciona como acompañamiento y/o base sobre la que se desarrollan distintas melodías simultáneas. Un único sonido nunca dará sensación de consonancia o disonancia. Para que éstas se produzcan hacen falta al menos tres sonidos distintos (acorde), el fundamental y otros dos armónicos. Melodía y armonía están totalmente interrelacionadas, pudiéndose considerar la melodía como una sucesión expresiva en el tiempo de sonidos pertenecientes a acordes armónicos, los cuales son enriquecidos con otros sonidos que adornan, suavizan, y producen efectos expresivos, complementando a los anteriores.

En el mundo de la música occidental, la armonía está relacionada con la “verticalidad” de la música, distinguiéndose de la línea melódica, o el aspecto “horizontal” de la misma.

El aspecto vertical mencionado tiene que ver con el uso simultáneo de diferentes tonos, como por ejemplo en los acordes. Desde hace varios siglos se descubrió que algunas combinaciones de acordes producen una sensación de tensión y tendencia al reposo. Algunos acordes, en un determinado contexto, tienen un sentido conclusivo y otros un sentido transitorio.

Esta idea no surgió durante la etapa griega, sino en la Edad Media. En ésta, el término armonía se utilizaba para designar el empleo de dos sonidos simultáneos, mientras que en el Renacimiento se le atribuyó este calificativo a la combinación de tres notas.

Las reglas de la armonía están a menudo sostenidas en las propiedades de la naturaleza, tales como ciertas relaciones numéricas descubiertas por Pitágoras, o las resonancias naturales. Pero estas reglas tienen más sentido conceptual que musical. La evolución de las reglas armónicas, en un contexto menos conceptual, surgen con la música religiosa de comienzos de la historia musical de occidente. Se ponen de relieve los intervalos paralelos perfectos. Estos intervalos debían preservar la claridad de la pieza original. Este trabajo se llevaba a cabo en catedrales, y hacían uso de los modos resonantes de sus respectivas catedrales para crear armonías. La utilización de estos intervalos paralelos se fue sustituyendo lentamente por el estilo inglés de consonancia, que usaba terceras y sextas. Se consideraba que este estilo tenía un sonido más dulce.

Se llama tercera a la tercera nota de la escala de cualquier tonalidad. El modo de la tercera puede ser mayor o menor, dependiendo de la distancia tonal que la separe de la tónica o primera nota de la escala. Si la distancia es de dos tonos, la tercera será mayor. En cambio, si están a tono y medio será menor. La sexta, por consiguiente, es la sexta nota de la escala. Si la distancia con la tónica, es de 4 tonos, la sexta es menor, mientras que si es de 4 tonos y medio es mayor.

La armonía tradicional de los estilos Barroco, Clásico, Romántico y parte del “Prebarroco”, es conocida como armonía tonal, ya que está basada en el sistema tonal, teniendo una fuerte función estructural (REYNOSO).

A partir del romanticismo musical, empieza a utilizarse con más fuerza el valor colorista de la armonía, debilitando paulatinamente la función estructural de la armonía tonal e introduciendo cada vez más modalismos, proceso que da lugar a la aparición de compositores impresionistas, nacionalistas y contemporáneos neoclásicos. Las músicas populares suelen utilizar armonías modales y muy características, como es el caso del flamenco, o armonías con una mayor componente tonal empleadas de forma sencilla, caso del tango.

2.2 Inteligencia artificial y música

Las primeras aplicaciones reales de la informática en la composición musical nacen casi a la par que los ordenadores.

Joseph Schillinger (1941), prediciendo el uso futuro de los ordenadores en la composición musical, intentó desarrollar una teoría musical que pudiese ser útil para su uso con cerebros electrónicos. No obstante el pionero de los trabajos llevados a cabo en el campo de la composición musical asistida por ordenador fue el químico y compositor Lejaren Hiller, que en 1955 inició sus trabajos de música algorítmica en la universidad de Illinois. En 1957, Hiller publicó la Suite Illiac (en homenaje al ordenador homónimo que la creó), primera composición musical íntegramente realizada por un ordenador, para la que combinó la utilización de cadenas de Markov (expuestas en 1905 por este matemático) y otros conceptos tomados de la teoría de la información, con sencillas normas musicales adaptadas del "Cantus Firmus" (canto dado habitualmente en redondas al que se añade el contrapunto o melodía realizada en la parte superior) (JORDÁ, 1990).

Durante el final de los cincuenta y gran parte de la década siguiente, la mayoría de los sistemas compositivos informáticos se vieron enormemente influidos por los trabajos de Shannon en la teoría de la información. Surgieron los primeros programas compositivos capaces de imitar un estilo musical prefijado. El de J.S.Bach (considerado como uno de los compositores más "algorítmicos"), fue (y sigue siendo) uno de los más trabajados. Con ligeras variaciones, estos sistemas solían combinar las frecuencias estadísticas de notas e intervalos (cadenas de Markov o entropía de orden 1 y 2), con medidas sobre la dispersión, probabilidad de ciertas formas, e intentaban incluir de algún modo datos sobre la estructura general (forma, repeticiones, etc.). Se realizaron varios experimentos que reunían a diversos compositores, instados a escribir de forma colectiva y por turnos, uno o dos compases en un mismo estilo musical predeterminado teniendo solo conocimiento de uno o dos de los compases inmediatamente anteriores, escritos por sus colegas.

Suele fecharse en 1968, año en que se publican dos artículos fundamentales, "Pattern in Music" de Herbert Simon y Richard Summer, de la Carnegie Mellon University, y "Linguistics and the Computer, Analysis of Tonal Harmony" de Terry Winograd del M.I.T., el inicio de la investigación moderna en inteligencia musical artificial. El artículo de Simon y Summer intenta explicar los patrones de la música tonal en términos de ritmo, melodía, armonía, y forma, como una extensión del formalismo empleado en los experimentos de procesado de información. Su intención era la de crear una herramienta para la comprensión de la actividad cognitiva en el oyente, que facilitase los estudios comparativos de estilo. Constituyó uno de los primeros acercamientos al todavía hoy confuso proceso de la cognición musical.

La investigación en inteligencia musical artificial se convierte así en uno de los paradigmas de la relación entre arte, ciencia y tecnología, y se establecen claramente sus dos principales vertientes: la científica, que estudia los procesos cognitivos, y la ingeniería o aplicada. Los modelos generativos posteriores, conformes con determinados idiomas musicales, no se limitarán a simular un estilo, sino que tratarán además de verificar la validez de diferentes

teorías musicales. Así por ejemplo, John Rothgeb consiguió demostrar la inconsistencia de la teoría del bajo continuo (acompañamiento que se improvisa sobre el bajo propiamente dicho y que da, sobre todo, a la música barroca un carácter muy personal) al tratar de hacerla computacional.

2.3 Inteligencia Artificial

Según la definición clásica, la Inteligencia Artificial (IA a partir de ahora a lo largo del trabajo) es el estudio de las facultades mentales a través del uso de modelos computacionales. Una tarea fundamental en la IA consiste en expresar a través de un lenguaje conveniente el conocimiento de la realidad de tal manera que pueda ser manipulado por una máquina inteligente y producir nuevos conocimientos. Un campo particular de la IA es el denominado Representación del Conocimiento (RC), cuyos objetivos son el desarrollo de modelos adecuados para la captura y manipulación del conocimiento.

Dentro de la IA hay diferentes teorías contrapuestas; por ejemplo, una teoría "lógica" expone que las manipulaciones deberían corresponder a mecanismos de inferencia en sistemas de lógica formal (representación del conocimiento de la realidad). En el lado contrario, la teoría "conexionista", más reciente que la anterior, afirma que las "máquinas inteligentes" deberían reflejar la estructura del cerebro y en las cuales el "conocimiento" no estaría expresado explícitamente en ningún tipo de lenguaje sino que sería "aprendido" por la máquina como estados-conexiones particulares entre unos relativamente simples elementos de procesamiento (neuronas).

Una aproximación tradicional a la IA asume que la actividad inteligente se obtiene a través de:

a) El uso de símbolos que representan el dominio del problema.
b) El uso de estos símbolos para generar soluciones potenciales a los problemas.
c) La selección de una solución adecuada al problema.

El uso de una técnica de representación del conocimiento adecuada es, por tanto, una de las claves más importantes para el diseño de sistemas de IA satisfactorios.

Asimismo, las matemáticas y la lógica, inherentemente ligadas a procesos musicales, juegan un papel fundamental en la formalización de lo que se considera inteligencia. La gran mayoría del trabajo de IA desarrollado hasta el momento asume que la inteligencia puede “simularse” encapsulando conjuntos de datos en paquetes estáticos de información. La actividad inteligente se lleva a cabo mediante un mecanismo que selecciona y combina paquetes apropiados de información almacenada en memoria para llegar a objetivos específicos.

En este sentido el conocimiento se clasifica en dos grupos:

a) Declarativo: por ejemplo la semántica de la gramática o el significado de los paquetes de información.
b) Procedimental: por ejemplo la gramática en sí misma o cómo ese mecanismo de tratamiento de la información debería funcionar.

El principal objetivo de las primeras investigaciones en IA fue reducir los procesos de razonamiento matemático a procesos de lógica formal. Las aproximaciones lógicas aplican los mecanismos de inferencia lógica a problemas de representación y razonamiento de la realidad. El problema es que este enfoque asume que los formalismos lógicos tienen éxito en dominios formales, tal como el de las matemáticas, pero ¿podrían jugar el mismo papel en dominios menos formalizados como por ejemplo el de la música?

Se arguye que las teorías simbólicas -lógicas y analógicas- carecen de plausibilidad cognitiva debido a que sólo existen acercamientos toscos a las actividades del cerebro. Otro enfoque ha sido bautizado como "conexionismo

eliminativo" dado que elimina el nivel simbólico. Diversos formalismos lógicos en esta línea, han sido explícitamente desarrollados o aplicados a la música.

Paralelamente a la IA existen los Sistemas de Inteligencia Artificial (SIA), que representan las implementaciones en ordenadores de modelos que se basan en hipótesis tal como las descritas anteriormente. Un programa informático basado en IA no tiene porqué ser un SIA. Por ejemplo, programas informáticos desarrollados mediante LISP o PROLOG –lenguajes típicos de programación en IA- no se deberían considerar como "inteligentes" tan sólo porque utilizan este tipo de lenguajes. Por el contrario, determinados programas informáticos que usan tipos de lenguaje procedimentales, tales como PASCAL o C, podrían ser SIA. De la misma forma, un sistema basado en técnicas de programación orientadas a objetos (POO) no es un SIA pues no tiene características "inteligentes", como por ejemplo la capacidad de inferencia, pero la POO ha demostrado que es una buena plataforma para el desarrollo de SIA.

Otro término habitual relativo a la IA es el de "sistema basado en el conocimiento". Un SIA no se convierte en un sistema basado en conocimiento tan sólo porque exista algo en el sistema que toma el conocimiento y puede escribirlo o expresarlo. Tampoco lo es por el hecho de que se comporte como si fuera inteligente. El SIA se convierte en un "sistema basado en el conocimiento" cuando la arquitectura contiene bases de conocimiento explícito. Para tener una base de conocimiento explícito el sistema debería contar con algún lenguaje bien especificado para codificar sus adquisiciones. Este papel es jugado por un lenguaje de representación del conocimiento.

Desde este enfoque se pueden identificar los tres elementos básicos para elaborar un sistema de representación del conocimiento:

- el lenguaje de representación
- el mecanismo de inferencia
- un ámbito específico de conocimiento

A continuación se hablará del subcampo de la IA en el que está basado este proyecto y por tanto el que aquí atañe, el sistema experto.

2.3.1 Sistema Experto

Un sistema experto es un sistema software que trata de reproducir el comportamiento de uno o más humanos "expertos" ante un problema específico (Wikipedia, 2008). Hay una amplia variedad de maneras para simular dicho comportamiento de los expertos, sin embargo, para que el sistema experto sea útil ha de tener las siguientes dos capacidades:

- *Base del conocimiento:* viene representada por un conjunto de reglas o pasos comprensibles de manera que se pueda generar la explicación para cada una de estas reglas, que a su vez se basan en hechos.
- *Motor de inferencia:* es un conjunto de mecanismos de razonamiento que permiten modificar los conocimientos anteriormente mencionados.

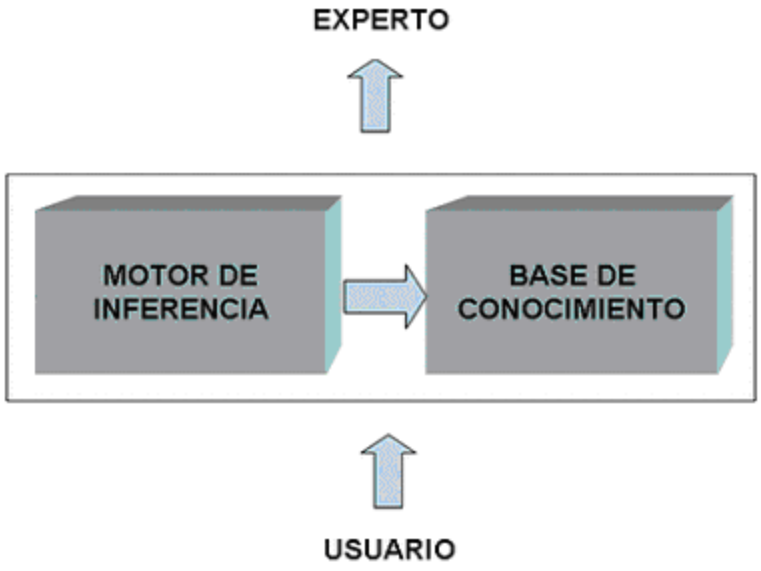


Figura 9: Arquitectura de un sistema experto

Una definición que aúna ambos conceptos se presenta a continuación.

Sistema Experto (DION, 2006) - sistema de información que representa el conocimiento de un experto dentro de un área particular de un problema como un conjunto de reglas, y lleva a cabo procesos de inferencia cuando se introducen nuevos datos.

Entre las aplicaciones originarias de la IA y la computación en cuanto a sistemas expertos se refiere, se encuentran la teoría de sistemas, operaciones de investigación, matemáticas aplicadas, etc.

Como ejemplo de aplicación de los sistemas expertos se puede mencionar el caso real de una refinería química en la que un empleado importante estaba a punto de retirarse, y la empresa veía en ello un grave problema puesto que era una pérdida muy importante de experiencia y conocimiento. Para solventar el problema se sustituyó al empleado por un sistema experto que reproducía su experiencia y conocimiento.

Las diferencias básicas entre un sistema experto y un programa ordinario se pueden ver en la siguiente figura.

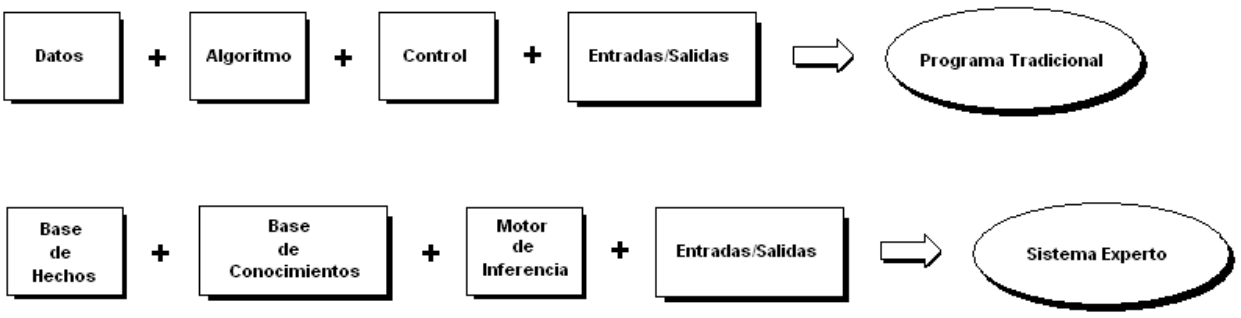


Figura 10: Flujos de información en los distintos sistemas

Existen varios tipos de sistemas expertos, pero los más destacables y extendidos son tres:

- *Basados en reglas:* trabajan mediante la aplicación de reglas, comparación de resultados y aplicación de las nuevas reglas basadas en situación modificada.

- *Basados en casos:* dan solución a determinadas situaciones nuevas basándose en las soluciones de problemas anteriores.

- *Basados en redes bayesianas:* modelo probabilístico multivariante que relaciona un conjunto de variables aleatorias mediante un grafo dirigido que indica explícitamente influencia causal. Su motor de actualización de probabilidades se basa en el Teorema de Bayes.

La forma más común en que se presenta un sistema experto es como un programa informático, con un conjunto de reglas (por lo general proporcionadas por el usuario del sistema), encargadas de analizar la información relativa a una clase específica de problemas y de proponer una o más acciones alternativas. El sistema experto podría también realizar un análisis matemático del problema. Un término relacionado con los sistemas expertos es “mago”, en inglés “wizard”. Un “mago” es un programa interactivo que ayuda a un usuario a resolver un problema. Originalmente este nombre se utilizó para programas que realizaban secuencias de búsqueda en bases de datos apoyándose para ello en las especificaciones del usuario. Sin embargo, algunos sistemas expertos basados en reglas se designan también con este término. Un sistema experto es muy eficaz cuando tiene que analizar una gran cantidad de información, interpretándola y proporcionando una recomendación a partir de la misma (SAMPER, 2009).

A continuación se va a explicar con más detenimiento cada una de las partes o capacidades que se mencionaron anteriormente y que constituyen un sistema experto.

Base del conocimiento

La base del conocimiento es una herramienta que pertenece tanto a la ciencia cognitiva como a la IA. En el campo de la ciencia cognitiva tiene que ver con cómo la gente almacena y procesa la información. En el de la IA, el principal objetivo es almacenar el conocimiento, de tal manera que los programas desarrollados computacionalmente puedan procesarlo y alcancen así la verosimilitud de la inteligencia humana. Las investigaciones hechas dentro del campo de la IA han tomado prestadas teorías de la ciencia cognitiva. Por tanto hay técnicas de representación tales como marcos, reglas y redes semánticas que proceden de las teorías del procesamiento de la información por parte de los humanos. Puesto que el conocimiento es utilizado para alcanzar un comportamiento inteligente, el propósito fundamental de la representación del conocimiento es representarlo de tal manera que facilite la inferencia, es decir, obtener conclusiones a partir del conocimiento.

Algunas de las cuestiones más importantes que surgen en la representación del conocimiento dentro de la perspectiva de la IA son:

- ¿Cómo representan el conocimiento las personas?
- ¿Cuál es la naturaleza del conocimiento y cómo se representa?
- ¿El esquema de representación debería tratarse como un campo particular o como propósito general?
- ¿Cómo de expresivo es un esquema de representación?
- ¿Debería ser el esquema declarativo o procedimental?

Motor de inferencia

Está relacionado con la representación elegida para el conocimiento del experto y se utiliza para procesar ese conocimiento. Se puede utilizar la componente de adquisición de conocimiento del sistema experto para introducir como entrada las múltiples características conocidas para conseguir una buena técnica de inferencia.

Para obtener la mencionada técnica de inferencia adecuada, hay que tener en cuenta las siguientes ideas:

- Una buena técnica de inferencia es independiente del ámbito del problema.

- Para encontrar las ventajas de la explicación, transparencia del conocimiento y la reutilización de los programas en un nuevo ámbito, el motor de inferencia no debe contener información específica del problema.

- Las técnicas de inferencia pueden ser específicas de una tarea particular, tal como un diagnóstico de configuración de hardware. Otras técnicas pueden ser asignadas solamente a una técnica particular de procesamiento.

- Las técnicas de inferencia son siempre específicas para la estructura del conocimiento en cuestión.

Ventajas y limitaciones

Las principales ventajas de los sistemas expertos son:

- Permanencia: a diferencia de una persona experta un sistema experto no envejece, y por tanto no sufre pérdida de facultades con el paso del tiempo, si bien habría que actualizarlo con el paso del tiempo.

- Duplicación: una vez programado, el sistema experto lo podemos duplicar infinidad de veces.

- Rapidez: un sistema experto puede obtener información de una base de datos y realizar cálculos numéricos mucho más rápido que cualquier ser humano.

- Bajo coste: a pesar de que el coste inicial pueda ser elevado, gracias a la capacidad de duplicación éste se reduce.

- Entornos peligrosos: el sistema experto puede trabajar en entornos peligrosos o dañinos para el ser humano.

- Fiabilidad: Los sistemas expertos no se ven afectados por condiciones externas, un humano sí (cansancio, presión, etc.).

Las limitaciones de estos sistemas:

- Sentido común: para un sistema experto no hay nada obvio. Por ejemplo, un sistema experto sobre medicina podría admitir que un hombre lleva 40 meses en estado, a no ser que se especifique que esto no es posible.

- Lenguaje natural: con un sistema experto no se puede mantener una conversación para comunicarse.

- Capacidad de aprendizaje: cualquier persona aprende con relativa facilidad de sus errores y de errores ajenos, mientras que para un sistema experto esto es muy complicado.

- Perspectiva global: un experto humano es capaz de distinguir cuáles son las cuestiones relevantes de un problema y separarlas de cuestiones secundarias. Un sistema experto tiene más dificultades para realizar esta acción.

- Capacidad sensorial: un sistema experto carece de sentidos.

- Flexibilidad: un humano es sumamente flexible a la hora de aceptar datos para la resolución de un problema. El sistema experto los necesita de una determinada manera.

- Conocimiento no estructurado: un sistema experto no es capaz de manejar conocimiento poco estructurado, hoy por hoy.

Tareas de un sistema experto

- Monitorización

La monitorización es un caso particular de la interpretación, y consiste en la comparación continua de los valores de las señales o datos de entrada y unos valores que actúan como criterios de normalidad o estándares. En el campo del mantenimiento predictivo los sistemas expertos se utilizan fundamentalmente como herramientas de diagnóstico. Se trata de que el programa pueda determinar en cada momento el estado de funcionamiento de sistemas complejos, anticipándose a los posibles incidentes que pudieran acontecer. Así, usando un modelo computacional del razonamiento de un experto humano, proporciona los mismos resultados que alcanzaría dicho experto.

- Diseño

El diseño es el proceso de especificar una descripción de un artefacto que satisface varias características desde un número de fuentes de conocimiento.

El diseño se concibe de distintas formas:

- En ingeniería es el uso de principios científicos, información técnica e imaginación en la definición de una estructura mecánica, máquina o sistema que ejecute funciones específicas con la máxima eficiencia.
- El diseño industrial busca rectificar las omisiones de la ingeniería. Es un intento consciente de traer forma y orden visual a la ingeniería de hardware donde la tecnología no provee estas características.

Los sistemas expertos en diseño ven este proceso como un problema de búsqueda de una solución óptima o adecuada. Las soluciones alternativas pueden ser conocidas de antemano o se pueden generar automáticamente probándose distintos diseños para verificar cuáles de ellos cumplen los requerimientos solicitados por el usuario, ésta técnica es llamada “generación y prueba”, por lo tanto estos sistemas expertos son llamados de selección. En áreas de aplicación, la prueba se termina cuando se encuentra la primera solución; sin embargo, existen problemas más complejos en los que el objetivo es encontrar la solución más óptima.

- Control

Un sistema de control participa en la realización de las tareas de interpretación, diagnóstico y reparación de forma secuencial. Con ello se consigue conducir o guiar un proceso o sistema. Los sistemas de control son complejos debido al número de funciones que deben manejar y el gran número de factores que deben considerar; esta complejidad creciente es otra de las razones que apuntan al uso del conocimiento, y por tanto de los sistemas expertos.

Cabe aclarar que los sistemas de control pueden ser en lazo abierto, si en el mismo la realimentación o el paso de un proceso a otro lo realiza el operador, o en lazo cerrado si no tiene que intervenir el operador en ninguna parte del mismo. Reparación, correcta o terapia. La reparación, corrección, terapia o tratamiento consiste en

la proposición de las acciones correctoras necesarias para la resolución de un problema. Los sistemas expertos en reparación tienen que cumplir diversos objetivos, como la reparación lo más rápida y económicamente posible, orden de las reparaciones cuando hay que realizar varias y evitar los efectos secundarios de la reparación, es decir, la aparición de nuevas averías por la reparación.

- Simulación

La simulación es una técnica consistente en crear modelos basados en hechos, observaciones e interpretaciones en el ordenador, a fin de estudiar el comportamiento de los mismos mediante la observación de las salidas para un conjunto de entradas. Las técnicas tradicionales de simulación requieren modelos matemáticos y lógicos que describen el comportamiento del sistema bajo estudio. El empleo de los sistemas expertos para la simulación viene motivado por la principal característica de los éstos, que es su capacidad para la simulación del comportamiento de un experto humano, lo cual es un proceso complejo.

En la aplicación de los sistemas expertos para simulación hay que diferenciar cinco configuraciones posibles:

- Un sistema experto puede disponer de un simulador con el fin de comprobar las soluciones y en su caso rectificar el proceso que sigue.
- Un sistema de simulación puede contener como parte del mismo a un sistema experto y por lo tanto éste no tiene que ser necesariamente de simulación.
- Un sistema experto puede controlar un proceso de simulación, es decir, que el modelo está en la base del conocimiento del sistema experto, y su evolución es función de la base de hechos, la base de conocimientos y el motor de inferencia, y no de un conjunto de ecuaciones aritmético-lógicas.

- Un sistema experto puede utilizarse como consejero del usuario y del sistema de simulación.
- Un sistema experto puede utilizarse como máscara o sistema frontal de un simulador con el fin de que el usuario reciba explicación y justificación de los procesos.

- Planificación

La planificación es la realización de planes o secuencias de acciones. Es un caso particular de la simulación. Está compuesto por un simulador y un sistema de control. El efecto final es la ordenación de un conjunto de acciones con el fin de conseguir un objetivo global. Los problemas que presenta la planificación mediante sistemas expertos son los siguientes:

- Existen consecuencias no previsibles, de forma que hay que explorar y explicar varios planes.
- Existen muchas consideraciones que deben ser valoradas o incluirles un factor de peso.
- Suelen existir interacciones entre planes de subobjetivos diversos, por lo que deben elegirse soluciones de compromiso.
- Trabajo frecuente con incertidumbre, pues la mayoría de los datos con los que se trabaja son más o menos probables pero no seguros.
- Es necesario hacer uso de fuentes diversas tales como bases de datos.

- Instrucción

Un sistema de instrucción realizará un seguimiento de un proceso de aprendizaje. El sistema detecta errores de una persona con conocimientos e identifica el remedio adecuado, es decir, desarrolla un plan de enseñanza que facilita el proceso de aprendizaje y la corrección de errores.

- Recuperación de información

  Los sistemas expertos, con su capacidad para combinar información y reglas de actuación, han sido vistos como una de las posibles soluciones al tratamiento y recuperación de información.

  Lo que diferencia a estos sistemas de un sistema tradicional de recuperación de información es que éstos últimos sólo son capaces de recuperar lo que existe explícitamente, mientras que un sistema experto debe ser capaz de generar información no explícita, razonando con los elementos que se le dan. Pero la capacidad de los sistemas expertos en el ámbito de la recuperación de la información no se limita a la simple recuperación. Pueden utilizarse para ayudar al usuario, en selección de recursos de información, en filtrado de respuestas, etc. Un sistema experto puede actuar como un intermediario inteligente que guía y apoya el trabajo del usuario final.

Proyectos desarrollados

En este apartado se van a presentar algunos de los lenguajes de programación que sirven para desarrollar programas que sean sistemas expertos, y algunos trabajos llevados a cabo en este subcampo de la IA:

- MYCIN: sistema experto desarrollado en 1972 para el diagnóstico de enfermedades infecciosas (GAE, 2009).
- CADUCEUS: sistema experto finalizado a mediados de 1980. Su propósito era mejorar el MYCIN. Utilizaba un motor de inferencia para tratar la complejidad adicional de las enfermedades internas.
- Dendral: es un sistema experto desarrollado a mediados de los años 60. Es el primer sistema experto en ser utilizado para propósitos reales. Su objetivo era interpretar la estructura molecular (ALONSO, 2001).
- Dipmeter Advisor: sistema experto cuyo propósito era ayudar al análisis de los datos acumulados durante la exploración de aceite.

Programas

A continuación se van a presentar algunos de los lenguajes de programación que sirven para desarrollar sistemas expertos:

- ART: es un lenguaje de programación de propósito general utilizado en el desarrollo de sistemas expertos.
- CLIPS: es un software de dominio público para construir sistemas expertos. Esta herramienta está basada en el lenguaje de programación C.
- Drools: es un motor de reglas de encadenamiento basado en inferencia. Es un sistema de producción de reglas que utiliza una mejora del algoritmo Rete.
- JESS: es un lenguaje de programación desarrollado por Java y basado en CLIPS para la construcción de sistemas expertos.
- Prolog: lenguaje de programación lógico e interpretativo para desarrollar aplicaciones en IA.

2.3.2 Enfoque estructuralista y búsqueda de una gramática musical

Parece lógico que al tratar de formalizar la música por primera vez se adopten modelos de la lógica formal y por extensión las matemáticas. Surge de forma casi simultánea en varios equipos de investigación la idea de la búsqueda de una gramática generativa para la música, que mediante un correcto alfabeto simbólico y un conjunto de reglas componga a partir de una descripción concisa de las ideas musicales fundamentales (JORDÁ, 1990).

A principios de siglo, Heinrich Schenker estableció un sistema transformacional de análisis musical. Schenker enfocó el problema desde un punto de vista estructural, dividiendo cada pieza musical en varios niveles (el orden cercano o superficial, que quedaba integrado por los intervalos existentes entre las notas, constituyendo los mínimos motivos musicales (de 2, 3 o 4 notas) el siguiente nivel, siendo el nivel más lejano la estructura profunda

o Ursatz, que describe la composición, y que engloba todas sus diferentes partes y sus repeticiones). Se trataba de un método en esencia analítico, no generativo.

Estudiando y aplicando las teorías schenkerianas destacamos a S. Smoliar, que en 1971 implementó en LISP varias de las ideas de Schenker, y consiguió generar polifonía medieval, canto gregoriano y contrapunto en forma de sonatas.

Las ideas de Schenker han permitido ampliar notablemente la comprensión de la estructura musical, relacionando y simultaneando sus aspectos melódicos y armónicos, pero su teoría permanece incompleta, imprecisa, y es todavía fuente de debate por parte de muchos musicólogos.

2.3.3 Problemas con la gramática

Teóricamente cualquier música susceptible de ser segmentada podría ser descrita por una gramática. En la práctica las gramáticas generativas han tenido sólo un éxito relativo en entornos compositivos limitados, como sistemas con estrategia particular, pero no han conseguido ofrecer soluciones o modelos generales. Dejando patente la falta de una teoría de la musicalidad sólida (JORDÁ, 1990).

Los modelos musicales desarrollados hasta ahora no han superado las dos dimensiones (tono y tiempo), evitando por consiguiente otros parámetros como la intensidad, y mucho menos el timbre.

Otra barrera a la que se enfrenta cualquier enfoque estructuralista de la música, es la de intentar plasmar la intuición musical en términos informáticos rigurosos. El intento de simular las habilidades composicionales humanas no resultará hasta que se consigan formalizar los planes de los músicos y sus efectos en los oyentes. Ian Morton ha desarrollado un modelo de percepción de

eventos musicales en términos de la complejidad de sus relaciones tonales, expresadas en forma de proporciones, pero muy pocos son los que han tratado de elaborar modelos de la actividad mental en el compositor.

2.3.4 Inteligencia artificial y conocimiento musical

Los intentos para modelar el conocimiento musical con inteligencia artificial dan lugar normalmente a un incremento de nuestro conocimiento de la psicología humana y el intelecto.

El conocimiento informático de la música implica cuatro problemas:

- Cómo se mide la música para proporcionar información de entrada al sistema informático.
- Cómo se presenta esa información al ordenador.
- Cómo esa información será representada dentro del programa informático para que dicho programa pueda llegar a algún entendimiento de su significado.
- Qué hará el ordenador con este conocimiento.

En primer lugar lo que se debe plantear para poder compatibilizar la música con la inteligencia artificial es qué se necesita medir, qué parámetros musicales hacen falta (DOBRIAN, 1993). La teoría musical de la música Occidental afirma que lo importante en ésta es entenderla como un conjunto de elementos simultáneos pero bien diferenciados: tono, duración, instrumentos, división... La manera de medir estos elementos tiene una gran dependencia de la cultura. Sin embargo, las unidades de medida pueden generalizarse en muchos casos.

Puesto que no se puede tener control de todos los parámetros existentes en una pieza musical, se debe decidir cuáles son los que se quiere medir. Una vez se han elegido éstos, lo siguiente que se ha de decidir es cómo se va a

hacer la medida de los mismos. Previamente a la medición se deben tener claros ciertos aspectos, como el grado de precisión al que se quiere llegar, qué se entiende por un resultado bueno o uno malo...

La selección de las cuestiones anteriores vendrá dada en base a la cantidad de información que se quiera suministrar al sistema, ya sea máxima o la mínima necesaria, lo que dependerá de cómo se plantea representar la información en el programa o qué se quiere hacer con ella. Por supuesto, a medida que la entrada de información aumenta, así lo hará también el potencial de detalle en la representación. Dependiendo de lo que se desee medir será bueno un nivel de detalle u otro.

Por ejemplo, si lo que se pretende evaluar es el tono y la duración de un fragmento musical, y la intención es reproducir el sonido original, sería deseable la máxima cantidad de información, mientras que si lo que se quiere es reproducir la notación, con un menor nivel de precisión sería suficiente.

Por norma general, es deseable que la medida de entrada tenga la mayor cantidad de detalle permitida por el sistema de representación, ya que el programa deducirá la información de manera algorítmica.

Sin embargo, en la práctica uno de los caminos más comunes en que los diseñadores de modelos cognitivos miden música es utilizando un controlador MIDI para capturar los datos mediante las “formas” del sonido.

A la hora de manejar los datos, uno de los problemas concretos que se encuentran es la percepción del ritmo.

Los seres humanos perciben el ritmo detectando patrones de eventos en el tiempo. Cualquier método de detección de estos patrones puede emplearse para procesar los datos musicales, para agruparlos en eventos musicales destinados a estar en el mismo patrón.

Los intervalos de tiempo dentro de los cuales se encuentran los patrones de este tipo de eventos se usan para estimar cuál será el ritmo percibido en una pieza de música, entonces dicho ritmo es analizado para obtener conceptos organizacionales tales como el pulso o la métrica.

El patrón más básico para analizar es la detección aislada de un evento cualquiera. Si, por ejemplo, se considera como evento la presión de una nota, se podrán realizar hipótesis sobre un ritmo basándoe en cadenas de intervalos de tiempo entre pulsaciones de notas. Al ser el más simple y obvio de los indicadores de ritmo es también el más usado.

Los intervalos entre teclas pulsadas (IOI – *Inter-Onset-Intervals*) se representan inicialmente como una cadena de números que muestran la medida en alguna unidad absoluta como pueden ser milisegundos. Esta cadena de números es lo que se llamará ritmo, pero para que tenga algún significado musical deben detectarse patrones en estas cadenas y el método que a priori parece más obvio para los músicos es tratar de comparar esta cadena de valores con ritmos conocidos de notación parecida. En caso de encontrar un ritmo parecido, los números de lo que se ha detectado pueden ajustarse hacia ese ritmo conocido y expresarse en términos relativos en lugar de absolutos.

Para determinar un ritmo con el que comparar, el oyente tiene que determinar primero un intervalo básico de tiempo que debe permanecer constante durante un cierto periodo. Se generará entonces el intervalo entre notas pulsadas para ajustarse a algún múltiplo entero o división del intervalo básico.

En la mayoría de las implementaciones informáticas la unidad de medida es considerablemente menor que el intervalo más pequeño que puede ser considerado un pulso musical. Por lo tanto los datos de entrada han de ser cuantificados. La mayoría de los secuenciadores MIDI utilizan un simple método de redondeo.

Las discrepancias que existan entre el ritmo procesado y el ritmo teórico serán debidas a tres factores:

- El primero de ellos son las pequeñas desviaciones que ocurren inevitablemente debidas al error motor. Se trata de un tipo de ruido matemático que puede ser considerado insignificante.

- El segundo factor puede denominarse error conceptual. Tocar una nota siempre está basado en una estimación del momento adecuado en el tiempo y en este proceso de estimación pueden ocurrir y ocurren errores.

Estos dos primeros errores forman parte de las desviaciones no intencionadas respecto a los valores teóricos.

- El tercer y último factor es el error intencionado, también conocido como rubato (consiste en acelerar o desacelerar ligeramente el tempo de una pieza a discreción del solista o el director de orquesta) o temporización expresiva, interacción entre la temporización y otros parámetros expresivos.

Estas últimas desviaciones pueden ser bastante más amplias que las anteriores, dependiendo en gran medida del estilo musical (por ejemplo, en la música Barroca las desviaciones en la temporización son más estrictas). Es probable que el rubato de un cantante a capella, por ejemplo, sea más exagerado que el de un cantante que está siendo acompañado por un patrón de arpegios regular.

Casi todos los detectores de ritmo trabajan sobre la idea de "expectativa", es decir, basándose en los ritmos percibidos hasta el momento, se hacen predicciones de los momentos temporales en el futuro en los que es probable que ocurran nuevos eventos rítmicos, reforzándose o debilitándose

estas predicciones a medida que el ritmo va coincidiendo o no con ellas. Estas alteraciones con respecto a las hipótesis harán que éstas vayan modificándose en favor de reconocer cada vez mejor los ritmos y melodías analizados.

Al margen de los intervalos entre pulsaciones o IOI's, hay muchos más factores que determinan nuestra percepción del ritmo, y que están disponibles para ser incluidos en un algoritmo de detección del ritmo, como por ejemplo los acentos dinámicos y la envolvente del tono que presentan todas las piezas musicales. Hay un parámetro rítmico en particular, la síncopa (desplazamiento del acento musical a notas normalmente no acentuadas), que es muy difícil de detectar por los sistemas computacionales dedicados a analizar el ritmo, y por lo tanto no se admiten como una posibilidad rítmica válida, lo cual limita en gran medida estilos musicales tales como el jazz, que utilizan a menudo ritmos sincopados.

Otro aspecto fundamental en la percepción de la música es la memoria musical. Se toman decisiones acerca de aquellas cosas que se consideran importantes para recordar, de ahí que se recuerden cosas que han ocurrido hace mucho tiempo, mejor que cosas que han ocurrido tan sólo hace unos momentos pero que son más prescindibles. La memoria selectiva es muy importante dentro de la percepción musical, por ejemplo, se recuerdan cosas importantes del comienzo de una pieza cuando reaparecen cerca del fin de la misma.

Por tanto se puede concluir que para poder integrar IA y música, hace falta tener muy claro qué factores musicales son importantes y de cuáles se puede prescindir, así como la manera en que se van a representar computacionalmente dichos parámetros, que constituyen el conocimiento musical.

2.3.5 Inteligencia artificial y composición musical

En este tema la mayoría de la información disponible se centra más en cómo se realizan estas actividades de composición musical y menos en por qué se hace o qué posibilidades tiene este campo.

La idea de que las decisiones pueden ser expresadas de manera algorítmica es la base de la IA, dado que los ordenadores sólo saben de cómo se hacen las cosas. Por tanto, el trabajo de los programadores de IA es precisamente convertir los “por qué” en “cómo”, lo que nos lleva a una discusión de problemas de toma de decisiones estéticas a la hora de usar ordenadores para componer música.

Lo primero que hay que hacer es especificar qué son las decisiones estéticas (DOBRIAN, 1993). Éstas son las que van encaminadas a la apariencia, es decir, las que se toman bajo criterios estéticos. Cuando el objetivo es la estética se busca que el resultado sea agradable, concepto harto ambiguo.

A la hora de componer, el compositor utilizará varios criterios estéticos, como por ejemplo a la hora de elegir el tono: la envolvente melódica, las implicaciones armónicas... aunque la elección no tiene porqué estar sujeta a criterios estéticos, sino que puede responder a sistemas preestablecidos. En este sentido, cuando se trata de componer música de manera metódica con el ordenador, el compositor simplemente seguiría reglas establecidas de toma de decisiones, algo que el ordenador haría mejor y más rápido que los humanos. Aun así, la existencia de esas reglas implica algunas decisiones estéticas.

Cuando un algoritmo es utilizado es porque el compositor decide con anterioridad que ese algoritmo debería conducir a resultados estéticos satisfactorios, por lo que se puede ver que la toma de decisiones basada en reglas puede siempre retroceder y encontrarse con alguna elección de prioridades, tanto estética como arbitraria. Si se considera que la decisión está basada en algún criterio estético que nunca podría ser comprendido intelectualmente, reconocemos una dimensión en la toma de decisiones

llamada a menudo “intuición”, sin embargo, si se acepta que una decisión estética puede reducirse a un punto en el que una elección supone mejoría respecto a otra se está introduciendo cierta aleatoriedad en el proceso, que puede ser la fuente de los resultados estéticos.

De todas formas no se conoce la manera para que un ordenador ejercite esas ideas de gusto o intuición, pero la aleatoriedad que se ha nombrado no supone ningún problema para él, lo cual normalmente es de escaso interés estético. Para que se produzca algo más que mero ruido blanco de aleatoriedad conocida, un programa informático de toma de decisiones estéticas debe contener algo no arbitrario, elecciones hechas por un programador, y aquí es donde surge una gran diferencia entre la música compuesta *con* un ordenador y la música compuesta *por* un ordenador, siendo hasta hace poco tiempo menos prioritaria la segunda corriente que la primera.

Sólo se puede decir que la música ha sido compuesta por un programa informático si es él quien toma realmente las decisiones. Estas decisiones pueden ser de carácter aleatorio, basadas en fundamentos de conocimiento de valores estéticos, o basadas en conocimiento adquirido (como en un sistema de Markov o de redes neuronales). Si no se incluyen elementos de decisión y simplemente el programa ejecuta una serie de reglas definidas, lo único que se demuestra es que el ordenador es una máquina de cálculo mejor que el ser humano, no que se esté comportando de manera inteligente.

El problema a la hora de intentar escribir un algoritmo de composición de uso general reside en que hay al menos tantas formas de hacerlo como compositores, y la mayoría de los compositores no aceptarán la idea de utilizar un algoritmo desarrollado por otra persona. Esto en principio significa que un compositor con ideas acerca de cómo emplear una máquina para componer, debe aprender a programar o colaborar con alguien que haga la tarea de programar los algoritmos específicos. Es muy difícil ser experto tanto en composición musical como en programación informática, por lo que la

colaboración entre músicos y programadores parece ser un buen camino para hacer música artificial.

La música tiene un cierto grado de jerarquía natural. Está compuesta por notas individuales que conectan las unas con las otras para formar frases, movimientos y piezas enteras. Sin embargo, otro de los problemas de estos algoritmos de composición es la falta de estructuras complejas o macroestructuras, puesto que las piezas simples de un único nivel jerárquico que dan lugar a múltiples melodías cortas sucesivas carecen de interés. Para esto se definen las gramáticas formales, que permiten crear y organizar estructuras jerárquicas. Estas gramáticas formales constan de un conjunto de reglas que expresan los símbolos de alto nivel, como puede ser la idea de "segundo movimiento", a través de una descripción de bajo nivel más detallada. Gracias a estas gramáticas formales el compositor puede elegir ahora una de aquellas macroestructuras mencionadas y producir mediante el algoritmo desarrollado una estructura más elaborada con la aplicación de la gramática formal.

Sistemas expertos y composición

Los sistemas expertos combinan el uso de conocimiento e inferencia, como ya se comentó en un apartado anterior, para tomar la siguiente decisión en cualquier tipo de problema, lo cual puede ser aplicado a la composición musical, puesto que ésta consiste en una sucesiva toma de decisiones, ya sean estéticas o basadas en reglas armónicas.

Como es natural, actualmente, la parte en la que el sistema experto puede ayudar a componer no es en la estética, si no en la regulada por normas, ya que las ideas que se pueden desarrollar en el conjunto que compone la base del conocimiento del sistema experto son de carácter fijo, pautado. La forma en que el cerebro percibe elementos y los clasifica según sus reglas naturales estéticas es muy compleja y por ello no puede ser implementada computacionalmente, hasta ahora.

El sistema experto puede deducir qué tono es más probable que venga a continuación en una pieza, a partir de un número variable de otros tonos anteriores, gracias a la parte denominada integrador del sistema.

Los sistemas expertos son los más aconsejables para componer música que pueda ser expresada como un conjunto de fundamentos, reglas y heurística, tales como el estilo de las obras para teclado de Bach.

La principal limitación de los sistemas expertos es que los nuevos estilos musicales no están tan bien definidos o no se han desarrollado lo suficiente como para ser codificados extensamente mediante las pautas anteriores. En estos casos el sistema debería ser capaz e aprender del material que se le facilita y completar la base del conocimiento por sí mismo.

3. DISEÑO DEL SISTEMA

3.1 Planteamiento del problema

Este proyecto constituye un sistema cuya funcionalidad es la de un compositor musical basado en IA. En apartados anteriores se han expuesto múltiples formas de abordar la problemática de la composición musical a través de la utilización de la IA. De todos ellos, para el desarrollo de este proyecto se decidió utilizar como herramienta el sistema experto, cuyos principios y fundamentos fueron expuestos en el capítulo anterior.

Realmente, la composición realizada por el sistema no consiste en una composición musical a partir de cero, sin ninguna clase de base, sino que se trata de construir un acompañamiento para una pieza ya creada con anterioridad y que esté constituida por una única voz, la cual le será facilitada al sistema de manera directa.

Se eligió el sistema experto porque permite describir y expresar este tipo de composición de manera muy semejante a como lo hace el cerebro humano. El sistema experto consta de dos partes, una, la base de conocimiento, que se asemeja a la memoria humana, donde se tienen almacenadas las reglas de armonía que permiten componer de una forma metódica más que estética, lo cual es más apropiado para nuestro proyecto puesto que la base estética de la melodía ya viene dada, y otra, el sistema de inferencia que le permite adaptarse al entorno, es decir, modificar, si fuese necesario, sus conocimientos.

Este proyecto no se ha centrado únicamente en la composición del acompañamiento melódico, sino que su función básica es la de integrar el conjunto de reglas (base del conocimiento), definidas mediante una herramienta de desarrollo de software de IA, con el esqueleto básico, desarrollado con un lenguaje de programación imperativa no perteneciente al conjunto de los lenguajes de IA, que se encarga de elegir qué regla utilizar en

función de las peticiones del usuario sin modificar ninguna parte de su código (las reglas utilizadas son independientes del código). Por esta razón la parte de la inferencia que compone el sistema experto no está desarrollada para que éste pueda aprender y modificar su base de conocimiento, sino que sirve para seleccionar qué regla de composición utilizar en cada momento.

El siguiente problema es la elección del lenguaje de IA a utilizar. Éste debe permitir definir de manera sencilla y clara las reglas para establecer la base del conocimiento, así como la integración fácil con un lenguaje de programación de propósito general, el cual también debe facilitar dicha tarea.

El lenguaje de IA que se ha seleccionado para el desarrollo de las reglas del sistema ha sido Jess, siendo Java la opción para el de propósito general. Se han seleccionado estos dos lenguajes ya que existe una librería en Java que permite interactuar fácilmente con la parte del sistema generada con Jess.

A continuación hay que decidir cómo se va a generar la melodía sobre la cual realizar el acompañamiento. Este problema queda fácilmente resuelto puesto que existe una librería en Java, llamada JMusic, que permite leer y manipular todos los parámetros útiles de un fichero MIDI, así como generarlos y editarlos.

Una vez solventados los problemas procedentes de la elección de las herramientas más adecuadas, vienen aquellos relacionados con la lógica, cómo desarrollar el programa para que interactúe con las reglas, qué reglas armónicas utilizar, puesto que existen cientos de ellas que regulan la escritura del acompañamiento, cómo implementarlas en Jess, cómo generar el esqueleto que las seleccione y qué parámetros son los necesarios para la realización de dicho esqueleto, cómo manipular los ficheros MIDI, etc.

La resolución de estos problemas será expuesta en los siguientes apartados.

Otro de los problemas surgidos viene a la hora de decidir qué tipo de melodías van a ser las que se permita introducir al sistema. Es decir, su complejidad, tonalidad, longitud, división del compás, etc. La elección de la estructura de las piezas de entrada se explicará con detenimiento en el apartado que trata sobre la generación del archivo MIDI.

3.2 Herramientas de desarrollo del sistema

Las herramientas que se han utilizado para la realización del sistema, son, una librería de disponible en la red llamada JMusic, desarrollada para Java, y un lenguaje de inteligencia artificial, Jess, el cual se integra fácilmente en el programa desarrollado a partir de otra librería implementada en Java.

A continuación se van a desarrollar con mayor profundidad y detalle cada una de estas herramientas.

3.2.1. Jess

Jess es un entorno script y un motor de reglas escrito enteramente en lenguaje Java por Ernest Friedman-Hill en Sandia National Laboratories en Livermore, CA.

Utilizando Jess se puede construir software Java que tenga la capacidad de “razonar” mediante el uso de conocimiento, implementado en forma de reglas declarativas. Provee de los medios necesarios para realizar la programación basada en reglas apropiada para desarrollar sistemas expertos. A menudo se le denomina como el *shell* de los sistemas expertos.

Este lenguaje destinado al desarrollo de inteligencia artificial es pequeño, ligero y uno de los motores de reglas más rápidos disponibles. Es un

lenguaje script potente que da acceso a todos los API's de Java, y fácilmente integrable con un programa en dicho lenguaje.

También incluye un entorno completamente adaptado para poder trabajar con la plataforma de desarrollo *Eclipse*.

Una gran ventaja disponible en este lenguaje es que incluye una versión mejorada y más robusta del algoritmo denominado *Rete*, cuya funcionalidad es la de procesar reglas.

El algoritmo *Rete*, mencionado en el párrafo anterior, es un mecanismo muy eficiente para resolver la dificultad del problema *many-to-many matching*, que se refiere al inconveniente existente a la hora de realizar la correspondencia entre múltiples patrones y múltiples patrones de objetos. Este mecanismo se basa en la utilización de lo que se denomina paradigma declarativo (*declarative paradigm*), el cual consiste en aplicar continuamente una colección de reglas a una colección de *facts*, mediante el proceso denominado *pattern matching*. Las reglas pueden modificar cualquier colección de *facts* o ejecutar cualquier código Java (JESS, 2009).

Estas reglas tienen una funcionalidad semejante a la de la sentencia condicional If-Then-Else en cualquier lenguaje de programación de alto nivel. Es decir, se podrían traducir en algo parecido a esto:

If (condición cumplida)
Then (realiza esto)
Else (realiza esto)

Por ejemplo, una posible regla en pseudocódigo podría ser la siguiente:

If (animal es un pato)
Then (el sonido es cuac)

La estructura de una regla es la siguiente:

```
(defrule rule-name
        "comentario opcional"
        (patrón-1)
        (patrón-2)
        …
        (patrón-n)
        =>
        (acción-1)
        (acción-2)
        …
        (acción-m)
)
```

Para que las reglas se ejecuten, es necesario que todos los patrones especificados en la primera parte de la regla (antes del '=>'), coincidan con la base de hechos, es decir, con los *facts* que estén definidos. Como se explicó anteriormente, el proceso seguido es el *pattern matching.* Los *facts* se pueden modificar, crear e incluso eliminar tras la ejecución de una regla (MENKEN, 2002).

La siguiente figura representa la interacción entre *facts* y reglas que se acaba de explicar:

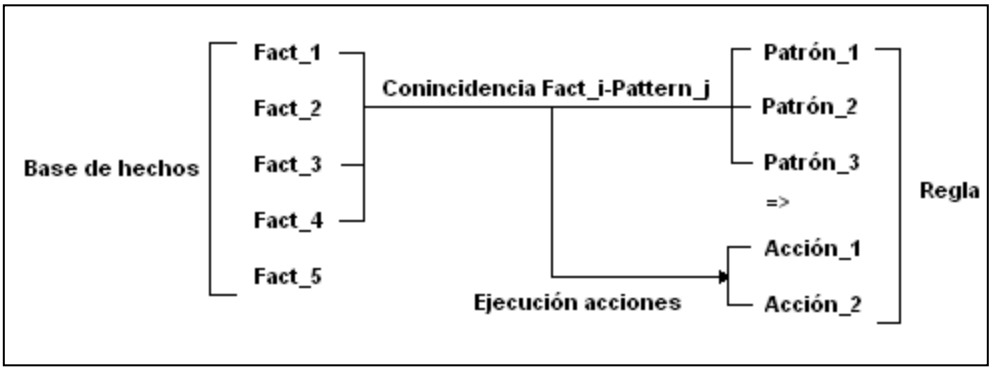


Figura 11: Mecanismo de actuación en Jess

La anterior figura simula la existencia de una base de hechos y una regla. La base de hechos está compuesta por cinco *facts*, y la regla tiene definidos tres patrones y dos acciones a realizar. Tres de los *facts* especificados en la base de hechos coinciden con los tres patrones definidos en la regla, lo cual desencadenaría la ejecución de las dos acciones.

Jess tiene definida una gran variedad de funciones semejantes a las existentes en los lenguajes de programación imperativa, tales como bucles (*while, for*…), condicionales (*if-then-else*…), comparación (*>, <...*), imprimir por pantalla, etc. Esto permite crear reglas con gran funcionalidad.

Jess dispone de una gran variedad de herramientas que facilitan el trabajo como las *memory querys*, y la manipulación directa de objetos Java. Con Jess, se pueden crear objetos Java de manera sencilla mediante instrucciones propias, llamar a métodos Java e implementar interfaces sin compilar ningún tipo de código Java, incluso se pueden generar applets y Servlets.

3.2.2 JMusic

Antes de empezar a describir con detenimiento el mecanismo de funcionamiento y las herramientas de las que consta JMusic, se va a realizar una breve introducción para presentarlo.

JMusic es un proyecto que ha sido diseñado para proveer a los compositores y desarrolladores de software de una librería compuesta por herramientas de procesado de audio y composición.

Constituye un sólido marco de trabajo para la composición asistida por ordenador en Java, siendo utilizada también para realizar música generativa, construir instrumentos musicales, reproducción interactiva y análisis musical.

Esta librería facilita a los músicos computacionales una sencilla estructura de datos musical basada en notas o eventos musicales, y provee métodos para organizar, manipular y analizar esos datos procedentes de composiciones musicales.

En JMusic los *scores*, o partituras, pueden ser traducidos a formato MIDI o ficheros de audio para su almacenamiento y posterior procesado, o pueden ser reproducidos en tiempo real. Con esta herramienta se pueden tanto leer como escribir ficheros MIDI, de audio, XML e incluso en su propio formato *.jm*. Consta de un servicio que permite el procesado en tiempo real para las aplicaciones JavaSound, QuickTime y MIDIShare (SORENSEN y BROWN, 1998).

JMusic ha sido diseñado para ser flexible, permitiendo al usuario generar código apoyado en su funcionalidad mediante la programación en Java, para crear sus propias composiciones musicales, herramientas e instrumentos.

Según afirman los creadores, la finalidad de JMusic es el apoyo y colaboración mutua, por ello se puede obtener de manera gratuita y es un proyecto de código abierto. Esta es, junto a su sencillez de manejo y su desarrollo en y para Java, la razón por la que se ha escogido esta herramienta para llevar a cabo el compositor.

A continuación se va a explicar en detalle cuál es la estructura de un fichero JMusic así como la funcionalidad de las partes de las que consta.

3.2.2.1 Estructura de datos de un fichero JMusic

La información musical se almacena en JMusic de manera jerárquica, basándose en la manera tradicional en que se hace en una partitura. Las diferentes partes o niveles que constituyen la estructura de la información con la que trabaja JMusic, se dividen en 4.

El primer nivel está conformado por el *score* (partitura), abarca toda la pieza o composición. El *score* se compone a su vez por las *parts* (partes), el segundo nivel, el cual actúa como un cajón en el que se almacenan las phrases (frases), tercer nivel, compuestas por un conjunto de *notes* (notas), cuarto y último nivel.

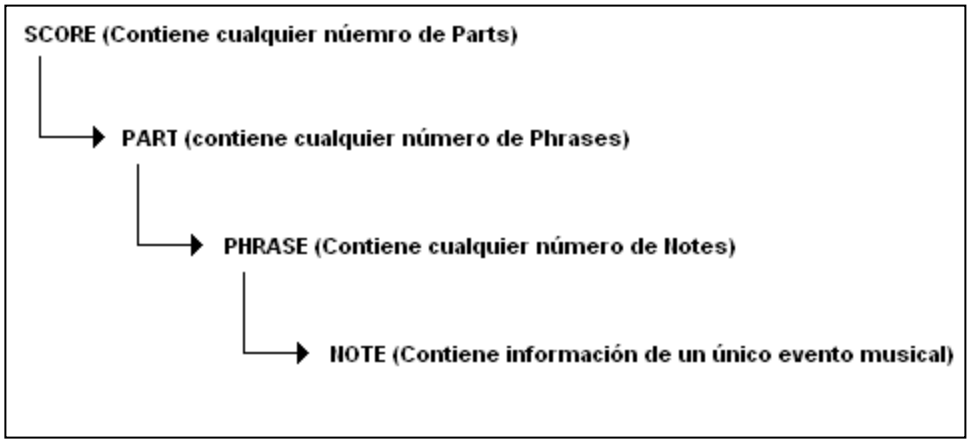


Figura 12: Estructura de datos en JMusic

Ahora se va a explicar parte por parte en qué consiste y de qué está compuesta.

3.2.2.2 Note (nota)

La clase *Note*, en el paquete *jm.music.data.Note*, es la estructura más básica utilizada por JMusic. Estos objetos están compuestos por múltiples atributos que contienen una gran cantidad útil de información musical sobre la nota. Los más destacables son los siguientes:

- *Pitch:* este atributo es el tono de la nota, es decir, la posición de la nota dentro de las diferentes escalas soportadas por JMusic.
- *Dynamic:* el volumen de la nota. Permite ajustar a cada nota un volumen diferente de manera que pueda tenerse en cuenta la expresividad de la obra, es decir, se pueden enfatizar o suavizar ciertas notas.
- *RythmValue:* el *RythmValue* hace referencia a la longitud de la nota en la obra. Esto es, si la nota tiene duración de negra, blanca, corchea, semicorchea, etc.
- *Pan:* es la posición de las notas dentro del espectro del estéreo.
- *Duration:* es la duración de la nota en milisegundos. Permite especificar de manera más concreta cuál es la duración de las notas a parte del *RythmValue*.
- *Offset:* es una desviación del comienzo "normal" en el tiempo de la nota.

La clase *Note* está compuesta también por un conjunto de métodos que permiten la manipulación sencilla de estos atributos para poder operar con ellos, y poder generar "pentagramas virtuales", es decir, poder escribir en el ordenador la pieza para más tarde traducirla a un formato de audio.

Tanto los tonos (*Pitch*) como la figura de la nota (*RythmValue*), vienen representados por un conjunto de constantes. En JMusic las constantes que representan los tonos vienen expresadas por una letra y un número. La letra indica cuál es la nota dentro de la escala en notación occidental estándar, y el número concreta en qué escala se encuentra la nota. Existen ocho escalas en JMusic, siendo la número cuatro la que se corresponde con la escala central en el piano.

C	D	E	F	G	A	B
Do	Re	Mi	Fa	Sol	La	Si

Tabla 1: Notación de notas occidental

Cada una de esas constantes tiene un valor numérico asociado. Este valor numérico es ascendente desde el Do bemol más grave, C0, hasta el Do natural más agudo, C9. La distancia establecida para semitonos es de una unidad, por tanto la distancia total desde una nota en una determinada escala, hasta su correspondiente octava mayor es de 12.

A continuación se muestra, como ejemplo de lo dicho, una tabla con los valores de la escala central y la siguiente, la 4 y la 5 respectivamente.

Nota	Valor Numérico	Nota	Valor Numérico
C4	60	C5/Bs4	72
Cs4/Df4	61	Cs5/Df5	73
D4	62	D5	74
Ds4/Ef4	63	Ds5/Ef5	75
E4/Ff4	64	E5/Ff5	76
F4/Es4	65	F5/Es5	77
Fs4/Gf4	66	Fs5/Gf5	78
G4	67	G5	79
Gs4/Af4	68	Gs5/Af5	80
A4	69	A5	81
As4/Bf4	70	As5/Bf5	82
B4/Cf5	71	B5/Cf6	83

Tabla 2: Ejemplo nota-valor numérico

En la notación empleada, la letra ‘f’ significa bemol, y la letra ‘s’ significa sostenido.

En la tabla aterior se puede ver lo explicado anteriormente, la distancia entre semitonos es de una unidad (Df4 = 61 || D4 = 62 || Ds4 = 63), y por lo tanto la distancia entre una nota y su correspondiente en una octava por encima, de 12 (C4 = 60 || C5 = 72).

Dentro de los tonos se encuentra el silencio, el cual viene representado por la constante REST (valor -2147483648).

La duración de la que se hablaba en párrafos anteriores, se refiere al atributo *RythmValue* en el contexto de JMusic, mientras que en música se refiere a la figuración, es decir, a las figuras representadas por negras, blancas, semicorcheas, fusas, etc.

Cada constante lleva asociado un valor numérico, al igual que sucedía con los tonos. Las figuras representan fracciones dentro de un compás. Por ejemplo, en un compás de cuatro por cuatro, la negra representa un cuarto de la duración completa del compás, mientras que la redonda representa la totalidad, el cuádruple, y la semicorchea un octavo, mitad de negra.

La correspondencia de tiempos es: redonda – blanca (mitad de redonda) – negra (mitad de blanca) – corchea (mitad de negra) – semicorchea (mitad de corchea) – fusa (mitad de semicorchea) – semifusa (mitad de fusa).

La distancia entre los valores numéricos de dos figuras contiguas en JMusic, es decir, que una dure la mitad que la otra, es de la mitad de la duración de la figura superior. Las constantes con las que se corresponden las figuras más comunes y sus valores numéricos asociados son las mostradas en la tabla siguiente.

Figura	Constante	Valor numérico

Redonda	WHOLE_NOTE	4
Blanca	HALF_NOTE	2
Negra	CROTCHET	1
Corchea	EIGHTH_NOTE	0.5
Semicorchea	SEMI_QUAVER	0.25
Fusa	THIRTYSECOND_NOTE	0.125

Tabla 3: Representación de la duración rítmica en JMusic

Las figuras con puntillo aumentan su duración la mitad de la duración de la figura original.

3.2.2.3 Phrase (frase)

Dentro de la estructura con la que trabaja JMusic, mencionada anteriormente, el siguiente nivel al de las notas lo constituye la frase.

Esta clase es un poco más compleja que la clase anterior, pero se puede explicar de manera muy sencilla en qué consiste, mediante una analogía con las voces que componen una pieza. Por ejemplo, el piano constituye una única parte dentro de la obra, sin embargo, ésta consta de varias voces, sin ir más lejos, puede llevar una voz en la mano derecha y otra en la izquierda.

Los objetos de la clase *Phrase* contienen un único atributo destacable, una lista de notas. Todo objeto de esta clase contiene una lista de objetos de la clase *Note*, pudiendo añadir nuevas notas a las ya existentes, eliminarlas o desplazarlas a otras posiciones dentro de la lista. Esta lista es un objeto de la clase *Vector* ubicada en el paquete *java.util.Vector* de Java.

Un conjunto de frases se pueden reproducir de manera secuencial, una detrás de otra, o en paralelo de tal forma que suenen a la vez, como voces superpuestas.

Una clase especial de frases son las *CPhrases*. Éstas constituyen una clase independiente de la clase *Phrase*. Las *CPhrases* permiten al compositor construir estructuras homofónicas de forma sencilla. Estas estructuras homofónicas son los acordes. Los acordes se pueden definir, de manera muy laxa, como un grupo de notas que suenan a la vez y duran el mismo tiempo, pero tienen diferentes tonos.

Las *CPhrases* son traducidas posteriormente a su creación, por JMusic, a *Phrases* simples. Tienen muchos métodos y características en común, sin embargo ahorran mucho trabajo a la hora de realizar obras relativamente complejas.

Mediante este tipo de frases se pueden generar secuencias de acordes sin necesidad de crear tres frases, una para cada nivel dentro del acorde, si se considera que éstos constan de 3 notas. Es decir, si se tiene la siguiente secuencia de acordes: Do-Mi-Sol || Re-Fa-La || Mi-Sol-Si, lo normal, si no se dispusiese de esta herramienta, sería generar un objeto *Phrase* para almacenar las tónicas (Do, Re, Mi), otro para las terceras (Mi, Fa, Sol) y otro para las quintas (Sol, La, Si), y disponerlos en paralelo para que sonasen todas las notas de un mismo acorde a la vez.

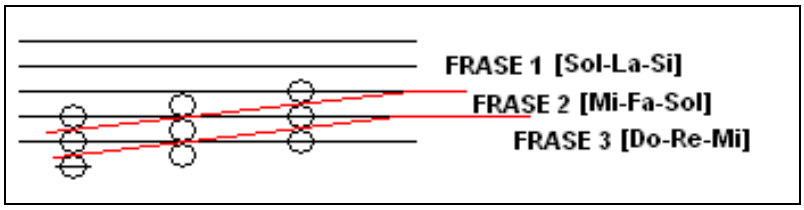


Figura 13: Ejemplo de creación de acordes con Phrase

Tanto los objetos *Phrase* como los *CPhrase* se agrupan en una estructura superior denominada *Part*, la cual será descrita a continuación.

3.2.2.4 Part (Parte)

Las partes constituyen el siguiente nivel dentro de la estructura del fichero JMusic. En ellas se introducen las frases generadas con las notas que constituyen la pieza.

Los objetos *Part*, contienen un vector que a su vez consta de múltiples frases, además de un canal y un instrumento, el cual viene representado en JMusic por un número y éste se corresponde con una posición en el array de instrumentos en audio.

Una parte, por tanto, será reproducida por un instrumento, pudiéndosele asociar uno dentro del array. De esta manera se puede simular la existencia de varios instrumentos sonando a la vez, y cada uno tocando la parte que le corresponda dentro del grupo.

3.2.2.5 Score (partitura)

Finalmente, se tiene el *Score*, que constituye el nivel superior dentro de la estructura datos, y está compuesta por un Vector de partes. El *Score* es el bloque que engloba todas las diferentes partes que se ha visto, para componer la pieza entera.

Esta clase consta de ciertos métodos que permiten abrir, crear o modificar ficheros de audio y MIDI. Al abrir un fichero y volcarlo en un objeto *Score*, se puede acceder a los distintos bloques que lo componen, como las partes, frases o notas. De esta manera se puede modificar libremente dicho fichero.

3.3 Arquitectura

En primer lugar, se va a plantear el diseño de la arquitectura del sistema completo, de qué bloques consta, para posteriormente explicarlos uno por uno con más detalle.

La arquitectura del compositor, viene fundamentalmente estructurada en base a la problemática surgida en el planteamiento del mismo y que ha sido expuesta en el primer apartado. El planteamiento de los problemas ha permitido proyectar y decidir cuáles son los bloques que hacen falta para poder abordar y resolver todo el sistema.

La representación más abstracta del sistema es la que se puede ver en el siguiente diagrama de bloques.

Figura 14: Diagrama de bloques general

El bloque de entrada/salida, engloba toda la funcionalidad que desarrolla las tareas de lectura de la melodía de entrada y de escritura del fichero de salida con la melodía armonizada.

El motor de reglas se encarga de realizar todas las actividades relacionadas con la manipulación de las notas, así como de preparar la información necesaria para la ejecución de la regla correspondiente y la recuperación del resultado procedente de ésta.

Las reglas constituyen la parte del sistema que se encarga de armonizar la secuencia de notas obtenida por el motor de reglas. Una vez armonizada, se devuelve la cadena de acordes de acompañamiento resultantes, al motor de reglas.

Como se puede ver, el sistema sigue un ciclo que comienza y termina en el bloque de entrada/salida. En primer lugar el bloque de entrada/salida lee la melodía a armonizar y se la entrega al motor de reglas. Éste la procesa y llama a las reglas tras haber generado las entradas correspondientes. Éstas se ejecutan y devuelven el resultado al bloque anterior, el cual lo manipula y lo

envía, de nuevo, al bloque de entrada/salida para que escriba el fichero final con la melodía armonizada.

En la figura siguiente se presenta un diagrama de bloques más detallado que el anterior. En él se muestra la estructura del sistema a más bajo nivel y la interacción y el orden de actuación de los distintos bloques.

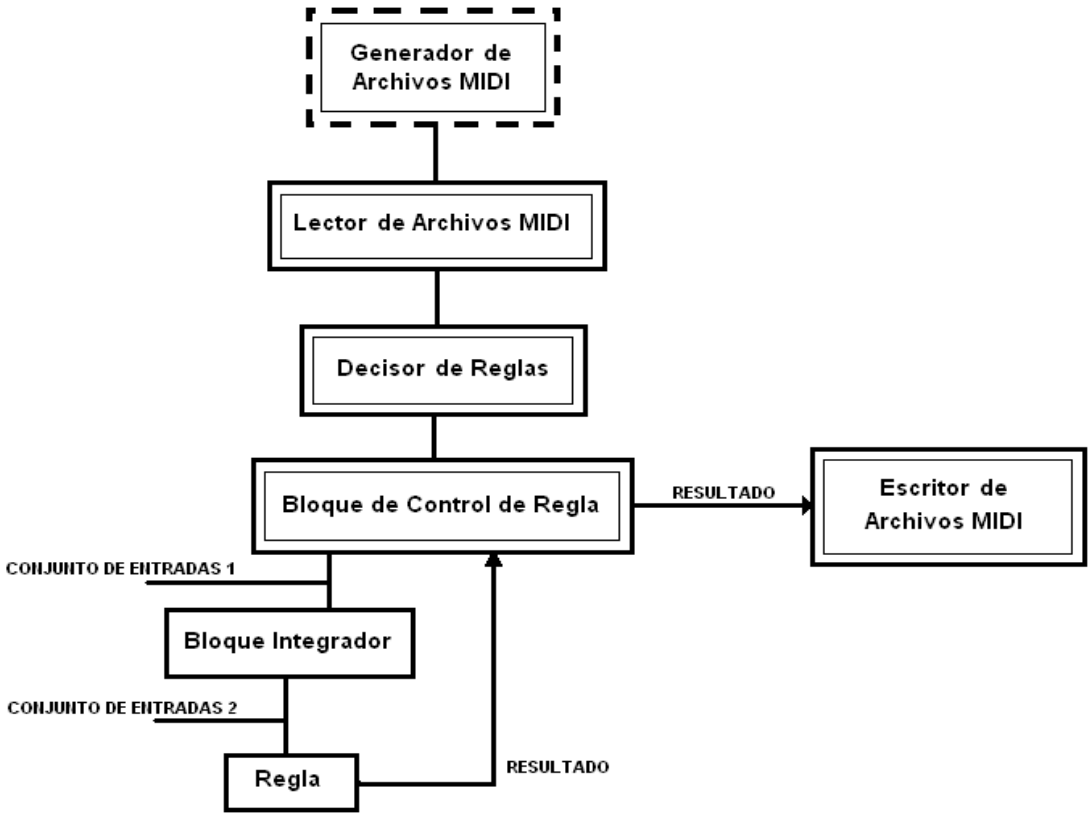


Figura 15: Diagrama de bloques del sistema completo

A continuación se van a explicar los distintos bloques representados en el diagrama anterior:

- *Generador de archivos MIDI:* no constituye un bloque del sistema como tal. Es un módulo auxiliar que permite crear las melodías de entrada al sistema.
- *Lector de archivos MIDI:* se encarga de leer el fichero MIDI generado con el bloque anterior, y devolver todas las notas que lo componen.

- *Decisor de reglas:* este bloque decide qué regla se va a ejecutar, en función de la opción elegida por el usuario.
- *Bloque de control de regla:* único para cada regla. Contiene toda la lógica necesaria para procesar los datos procedentes del fichero MIDI, en función de la regla que se utilizará, y prepararlos para que puedan ser interpretados por el lenguaje de inteligencia artificial.
- *Bloque integrador:* también es único para cada regla y puede considerarse un sub-bloque del controlador. Se encarga de preparar toda la información necesaria para la ejecución de la regla, así como de cargar y ejecutar la regla.
- *Regla:* es el código implementado en lenguaje de inteligencia artificial que representa la regla de armonía que se ejecutará.
- *Escritor de archivos MIDI:* este bloque genera la salida del sistema completo en formato MIDI. Escribe el fichero MIDI con la información procedente del bloque integrador, que se supone será ya la pieza completa, primera voz y acompañamiento.

Una vez descrita la arquitectura del sistema, y presentados los distintos bloques que lo componen, se va a proceder a describirlos de manera más detallada.

3.4 Generador de archivos MIDI

Como ya se ha comentado, no constituye un bloque funcional dentro del compositor, sino que más bien es un módulo auxiliar que se encarga de generar las melodías de entrada. Escribe el archivo MIDI que contiene la pieza de una única voz, sobre la que se ejecutará el compositor para generar toda la línea del acompañamiento.

3.5 Lector de archivos MIDI

Este bloque se encarga de realizar la lectura la composición de entrada a armonizar. La herramienta utilizada para ello es la librería desarrollada para Java, JMusic, que permite acceder y procesar de manera muy sencilla ficheros MIDI.

Tener una estructura del archivo MIDI definida y específica a la hora de generarlo es importante, ya que en este punto se ha de acceder a él para obtener los distintos parámetros, como las notas, ritmos, etc., necesarios para introducirlos como entradas del compositor.

El MIDI se realiza mediante un programa sencillo, escrito con JMusic, que consta de una única clase y un método *main*.

Las melodías se escriben de forma directa en el *main*, nota a nota, respetando la nomenclatura de la librería, para posteriormente ser empaquetadas en un fichero MIDI.

La pieza de entrada al compositor tendrá siempre la misma estructura básica y será sencilla, puesto que el objetivo principal del proyecto no es crear un compositor capaz de generar acompañamientos de obras enormes, sino de realizar un sistema experto capaz de integrar el esqueleto básico que selecciona las reglas, definido por un lenguaje de propósito general, y las reglas descritas por el lenguaje de inteligencia artificial.

La línea melódica vendrá compuesta por una única voz, y su longitud será suficiente como para permitir comprobar el correcto funcionamiento del sistema.

A su vez, esta voz no puede ser muy compleja en cuanto al ritmo se refiere, puesto que las reglas aplicadas, en la actual versión, no están orientadas a realizar un análisis exhaustivo del ritmo y de la musicalidad, sino que buscan generar una frase cuya melodía concuerde con la original. Esto es así puesto que el análisis del ritmo no es una tarea sencilla para un ordenador, sobretodo si se entra a estudiar ritmos complicados y poco homogéneos.

Esta complejidad del ritmo de la que se habla, se refiere fundamentalmente a la aparición de silencios o la variación muy rápida de las figuras que lo componen. La variación rápida no se ha de confundir con la velocidad que marca la subdivisión (corchea, semicorchea, etc.), puesto que hace referencia a la figuración, al cambio más o menos frecuente de las formas rítmicas. Los silencios entran dentro de esta restricción ya que también generan una alteración en el patrón rítmico de la pieza, y esa alteración puede ser provocada por un simple silencio de negra, el cual se tendría fácilmente en cuenta en un compás cuya división básica sea un múltiplo entero de negra, o por algo más complicado, como por ejemplo, un silencio de semicorchea con puntillo, el cual da lugar a un patrón en el que hay que tener en cuenta sucesos de notas en fracciones del tiempo más complejas.

La tonalidad es otro de los parámetros destacables en este proyecto, puesto que toda melodía, ya sea sencilla o muy complicada, lleva asociada una tonalidad. La tonalidad marca el carácter de la pieza (alegre, triste, solemne, lúgubre, etc.), así como el conjunto de notas que pueden intervenir en ella.

Este parámetro viene dado por el usuario del compositor, es decir, no es uno de los factores que éste tenga que obtener, como pueda ser el ritmo o la melodía en sí. La obtención de la tonalidad es un problema que podría ser el objetivo de un módulo adicional para este proyecto, siendo por ahora innecesario para que éste desarrolle su función adecuadamente.

En un primer lugar se pensó el compositor para trabajar solamente en una tonalidad, Do Mayor, sin embargo, después se generalizó para que pudiese trabajar en cualquiera.

La tonalidad es un factor importante a tener en cuenta, puesto que marca cuál es la nota dominante, hecho fundamental a partir del cual se empiezan a construir los acordes que servirán para generar el acompañamiento. Su utilización y funciones se describirán con más detalle en

el siguiente apartado (implementación), cuando se describa punto por punto la funcionalidad del sistema desarrollado.

Otro de los factores importantes dentro de la línea melódica de entrada, es el modo de la tonalidad, es decir, si es mayor o menor. Éste va intrínseco en la tonalidad, sin embargo, hay que especificarlo aparte como una entrada del sistema. Su utilidad también se detallará en el capítulo siguiente.

La información de la tonalidad y su modo, no va dentro del fichero MIDI de entrada, sino que se especifica aparte en una variable de entrada al sistema. Se ha hablado de ellas, porque aunque no hagan falta para generar el MIDI, sí son necesarias para escribir la pieza de entrada, puesto que toda pieza se crea sobre una tonalidad.

La morfología del fichero generado es también importante especificarla, para que en el momento en que sea leído por el bloque siguiente, éste conozca su estructura.

Esta morfología es simple puesto que, como ya se ha dicho, la melodía sólo se compone de una voz. Su estructura completa (a nivel de fichero MIDI) será definida cuando se hable, en el apartado siguiente, del programa con el que se genera, JMusic, y en concreto del bloque lógico correspondiente.

En las siguientes figuras se muestran un par de ejemplos del tipo de pieza de entrada que se genera en este bloque:

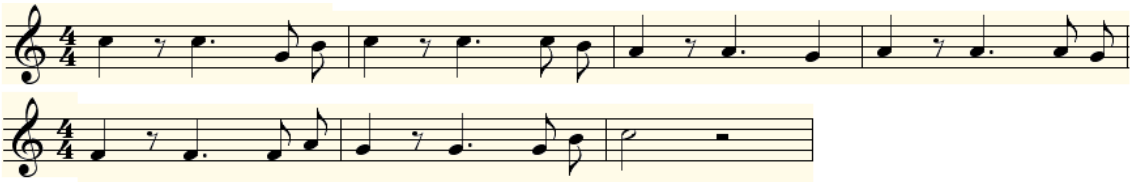


Figura 16: "Stand by me", por Ben E. King

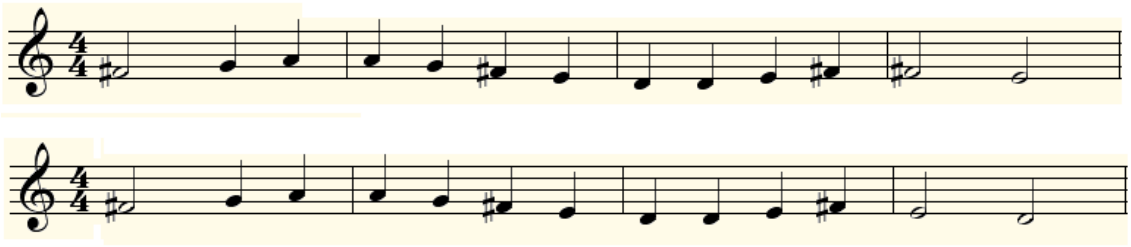


Figura 17: "Himno de la Alegría", por Ludwig van Beethoven

3.6 Decisor de regla

Como ya se ha dicho en apartados anteriores, el compositor creado en este proyecto ha sido desarrollado como un sistema experto, los cuales están compuestos por la base de conocimiento y el motor de inferencia. La base del conocimiento vendría dada por un conjunto de reglas que dictan las acciones a realizarse en función de lo que se desee hacer, al igual que ocurre en el cerebro humano.

En este caso, la base del conocimiento está compuesta por una regla constituida por otras tres, las cuales se complementan para dar una funcionalidad completa a la regla principal.

El decisor de reglas, implementado en lenguaje de propósito general, se encarga de seleccionar qué regla será aplicada en cada momento, suponiendo que se tuviese un gran conjunto de éstas. La lógica que compone este bloque no varía a la hora de seleccionar la regla que se ha de aplicar para obtener el resultado, es independiente de ellas.

3.7 Bloque de control de regla

Su función es preparar la información para que la regla en cuestión pueda ejecutarse correctamente y manipular el resultado obtenido de manera que pueda ser interpretado y utilizado por el resto del sistema.

La lógica que se implementa en el bloque de control, se realiza con un lenguaje de propósito general (Java) en lugar de uno de inteligencia artificial. Esto se debe a que éste tipo de lenguajes son mucho más limitados que uno de propósito general e impiden realizar ciertas funciones con la misma facilidad y claridad que los otros. En el caso de este sistema, se tiene que manipular información antes y después de la regla, de tal modo que ésta quede integrada en un entorno que en principio no es adecuado para ella, por ello es necesario tener a nuestra disposición dichas funcionalidades.

El bloque controlador es único para cada regla ya que cada una tiene una estructura y función diferentes. Por tanto, no pertenece a la parte del sistema que permanece constante y que es independiente de la regla que se ejecute. Cada una necesita la información de entrada de una manera concreta.

La información que ha de procesar el bloque de control antes de que se ejecute la regla son, básicamente, los parámetros extraídos por el lector de archivos MIDI a partir del fichero musical procedente del generador de melodías. El controlador de regla, conoce cuál es la estructura del fichero MIDI y la forma en que es extraída la información musical que contiene.

Una vez obtenida esta información la utilizará y procesará para generar parte de las entradas necesarias para la ejecución de la regla.

El siguiente paso que debe llevar a cabo el controlador es ejecutar el integrador del sistema, el cual constituye un sub-bloque de éste, para que termine de preparar todas las entradas necesarias para la ejecución de la regla en cuestión.

El resultado que genere la regla una vez haya terminado de ejecutarse, es recogido por el bloque controlador, que lo termina de procesar y lo envía al siguiente bloque para que concluya el proceso.

3.8 Bloque integrador

Éste se puede considerar un sub-bloque del controlador, ya que su función es completar el conjunto de entradas necesario para ejecutar la regla que se quiera utilizar, y se encuentra dentro del ciclo de ejecución de la parte de control.

El integrador también es único para cada regla, ya que se encarga de especificar y generar todas las entradas restantes, que el controlador no ha realizado. Define el estado inicial de la ejecución de la regla, a partir del cual se desencadenará todo el proceso de análisis de ésta dando lugar al resultado final.

Este bloque se encarga también de llamar a la regla una vez ha establecido todos los parámetros iniciales.

3.9 Regla

La regla es el motor fundamental del compositor, cuya ejecución sólo es posible tras una serie de preparativos, para obtener el resultado casi final.

Las reglas definidas para este proyecto, y las que se puedan definir en adelante para completar y dar una mayor funcionalidad al compositor, constituyen la base del conocimiento del sistema experto. En ellas reside la actividad principal de toda la cadena de acciones, y son ellas, junto a sus respectivos bloques de control, las que dan el carácter "inteligente" al conjunto total que compone el sistema.

La regla que se ha utilizado en este proyecto, pertenece al amplio conjunto de reglas armónicas definido en la música. Éstas se han simplificado, puesto que existen muchísimas reglas en los tratados de armonía para componer una única línea melódica que sirva como acompañamiento.

Este bloque se ha desarrollado mediante un lenguaje destinado a la inteligencia artificial llamado Jess. Se podría decir que la regla completa está constituida también, por el controlador de la regla. Sin embargo, en ésta es en la que reside la verdadera teoría armónica, mientras que en el sistema de control se llevan a cabo más funciones de preparación.

La regla definida en este sistema compositor, establece qué notas pueden combinarse de forma que constituyan un conjunto en equilibrio armónico. Lo que realmente especifica es el grado sobre el cual hay que construir el acorde, dentro de la tonalidad en la que nos encontremos, para la nota actual que se esté analizando.

El bloque de control agrupa una serie de notas bajo un determinado criterio, que se especificará a la hora de describir el funcionamiento del programa, y a ese conjunto de notas se les asigna un acorde teniendo en cuenta la primera nota del conjunto. Este proceso se repite sucesivamente hasta el final de la pieza.

Existen múltiples combinaciones a realizar entre las notas dentro de una tonalidad. La combinación de acordes por los que se ha optado, para asignar a cada nota de la tonalidad no es única. En la siguiente tabla se presenta esta relación de notas para cualquier tonalidad.

Grado	Acorde adecuado	Acorde reemplazante
I	I	IV
II	V	IV
III	I	IV
IV	IV	II
V	V	III
VI	IV	II
VII	V	VI

Tabla 4: Correspondencia de acordes principales y de acompañamiento

El acorde adecuado constituye el acorde que habría que aplicar a la nota en cuestión en primera instancia. En caso de que el acorde asignado a la nota que se esté analizando en ese momento, se hubiese utilizado justo en la nota anterior, se emplea el acorde reemplazante.

A continuación se muestra en una tabla cuál sería esta correspondencia de acordes para la tonalidad de Do mayor.

Nota	Acorde adecuado	Acorde reemplazante
C	C	F
D	G	F
E	C	F
F	F	Dm
G	G	Em
A	F	Dm
B	G	Am

Tabla 5: Ejemplo de correspondencia de acordes

En esta tabla se ha sustituido la columna *Grado* por *Nota*, ya que se han sustituido los grados por las notas específicas con los que se corresponden en la tonalidad de Do mayor. La letra *m* al lado del acorde, representado por una letra mayúscula, significa que éste es menor.

La salida generada es una cadena formada por una sucesión de números que representan los grados sobre los que hay que construir el acorde correspondiente dentro de la tonalidad. Estos grados serán interpretados por el controlador de la regla que generará una serie de notas para que el siguiente bloque pueda generar el fichero MIDI final.

3.10 Escritor de archivos MIDI

Este es el último bloque del sistema compositor. Su objetivo es generar el fichero MIDI de salida con las modificaciones que se hayan realizado durante el proceso de composición.

No sobrescribe el archivo original, sino que genera uno nuevo a partir de éste, incluyendo la parte nueva. El nuevo archivo tendrá el mismo nombre que el original, añadiéndole la palabra “Nuevo”.

3.11 Diagrama de flujo

En este apartado se van a explicar uno por uno los pasos seguidos por el sistema a lo largo de su ejecución. Para mayor claridad, se han dibujado cuatro diagramas de flujo.

El primero muestra el proceso completo de manera general, sin entrar en detalles.

En el segundo se explica el funcionamiento de la parte más compleja del sistema. Se puede ver con detalle todo el proceso que sigue el compositor para dividir la pieza en grupos y asignarles un acorde a cada uno.

Con el tercer diagrama de flujo se presenta la última parte del proceso, compuesta por unas pocas actividades que sirven para finalizar y generar el archivo MIDI de salida.

Y el cuarto representa la ejecución de la regla.

A continuación se muestra el primero de los diagramas de flujo comentados.

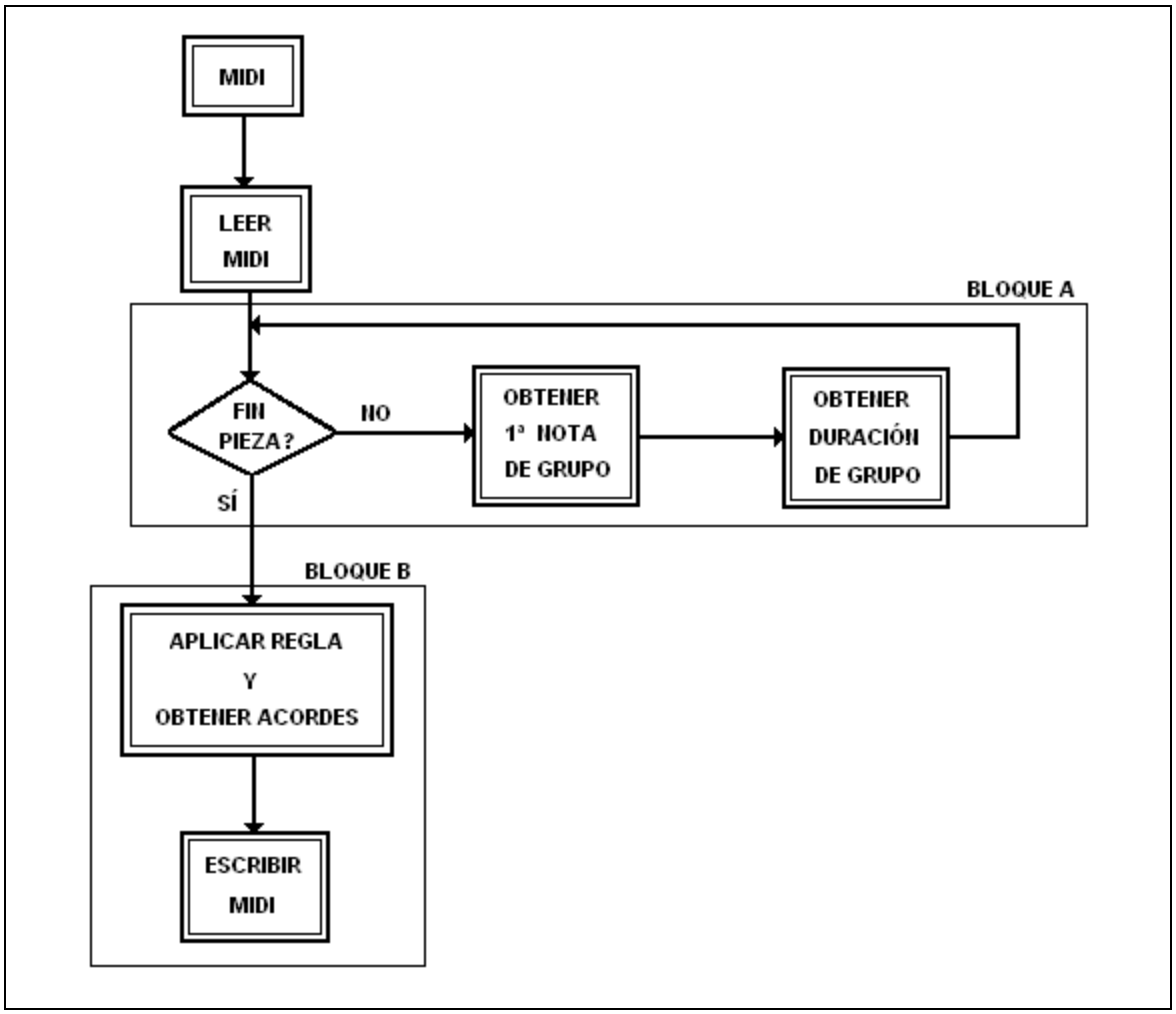


Figura 18: Diagrama de flujo del sistema completo

En él se puede ver cuáles son, de manera general, los pasos que sigue el compositor para generar el fichero final con el acompañamiento realizado para la melodía de entrada.

El primer paso es leer el fichero MIDI con la melodía de entrada, para poder procesarla y generar el acompañamiento. Llegados a este punto se extraen las partes que componen el fichero. En este caso, el fichero estará compuesto por una única parte (*Part*), como ya se dijo en apartados anteriores, y tendrá una frase que estará constituida por la secuencia de notas que da lugar a la melodía principal.

Una vez se ha descompuesto el fichero y extraído una lista con las notas de la pieza, comienza el análisis de la misma. Se recorre la lista nota por nota

para crear grupos, o frases musicales (que no tienen que ver con las *Phrase* de JMusic), de una longitud musical máxima modificable.

El usuario puede modificar la longitud máxima de los grupos, en caso de que la que se haya especificado en principio o por defecto no dé buen resultado en la composición. Esta longitud se especifica en duración de redonda, blanca, negra, etc., en base al ritmo de la pieza.

Los grupos se forman obteniendo y almacenado en un Vector la primera nota del mismo, y en otro de la misma longitud, la duración de las notas adyacentes que se incluyen en el grupo.

El proceso de selección de las notas para la formación de dichos grupos está representado en el siguiente diagrama de flujo (bloque A en el diagrama anterior).

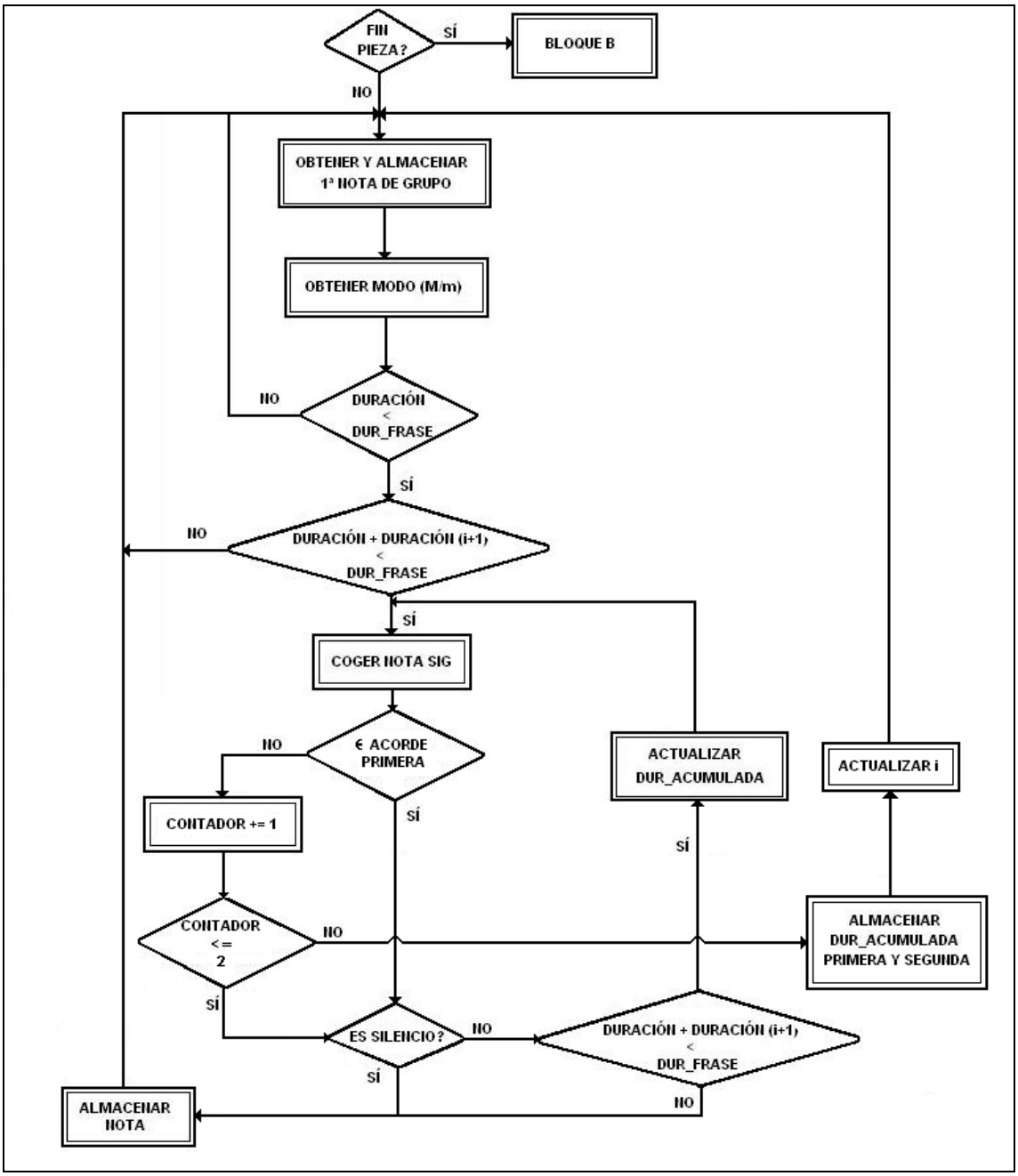


Figura 19: Diagrama de flujo A

El primer paso, es comprobar si se ha alcanzado el final de la pieza, en cuyo caso se daría paso a la ejecución del *Bloque B*. En caso contrario comienza la división de la melodía en conjuntos.

Esta comprobación de fin de pieza se realiza al principio del proceso completo y después de haber finalizado la creación de un grupo. Por ello, la

primera nota que se obtenga después de la comprobación será primera nota de grupo.

Es necesario almacenar las primeras notas de cada grupo, porque como se verá más adelante, la decisión del acorde que se establecerá en el acompañamiento para un grupo concreto se llevará a cabo sobre ellas.

Después de haber almacenado el tono y la duración de la primera nota del grupo, se obtiene el modo mayor o menor, correspondiente al acorde de dicha nota en la tonalidad que haya sido especificada. Esto es, si la primera nota de grupo es Re, y la tonalidad es Do mayor, el modo del acorde de Re en Do mayor es menor. Esto es necesario porque unos pasos más adelante se utilizará esta información para comprobar si las notas siguientes pertenecen o no al acorde de la primera.

A continuación se comprueba si la duración de la nota es superior a la duración máxima de la frase, en cuyo caso se toma la nota actual como un grupo y se vuelve al comienzo del proceso analizando la siguiente nota de igual manera que se ha analizado la anterior.

Si la nota no superó la duración máxima, se comprueba si su duración más la duración de la siguiente sí lo hace. En tal caso, se realiza el mismo proceso que en el caso anterior.

En caso de que ni la duración individual de la nota que se está analizando, ni la duración conjunta con la siguiente nota superen la duración máxima de la frase se pasa a analizar la siguiente nota. Una vez se ha obtenido ésta, se verifica si pertenece al acorde de la primera nota del grupo. En caso de que no sea así, se incrementa en una unidad un contador que contabiliza qué notas, de las que se van analizando, no pertenecen a dicho acorde.

Seguidamente se lleva a cabo una comprobación consistente en ver si es un silencio. Si es un silencio se almacena su duración y su tono como se hizo con la primera nota de grupo y retornamos al comienzo.

Si no es un silencio, se guarda su duración sumándose a la que ya había acumulada (que era la de la primera nota del grupo). A continuación se verifica si la duración acumulada más la duración de la nota siguiente (que sería la tercera nota analizada desde el comienzo de la formación del grupo), superan la duración máxima de la frase. En caso de que sí la superen, se almacena la duración acumulada de ambas notas y se vuelve al comienzo.

En caso contrario, se almacena la duración acumulada y se pasa a analizar la siguiente nota.

Este proceso se repite hasta que la duración acumulada de las notas del grupo más la de la siguiente nota que se va a analizar sea mayor que la duración máxima de la frase, o hasta que el contador de notas que no pertenecen al acorde de la primera de grupo llega a tres. En este caso, se rompe el grupo y nos quedamos únicamente con la primera y la siguiente a la primera. Después se comienza el proceso a partir de la tercera nota que componía el grupo. Esto es debido a que más de tres notas que no pertenecen al acorde que las acompaña suena amelódico. Este proceso se repetirá hasta que se termine con todas las notas.

Tras formar el grupo y almacenar la duración acumulada de las notas que lo componen, se continúa este proceso hasta que se alcanza el final de la pieza.

Una vez formados todos los grupos, y extraídas todas las primeras notas que los componen, tiene lugar el proceso descrito por el diagrama de flujo B.

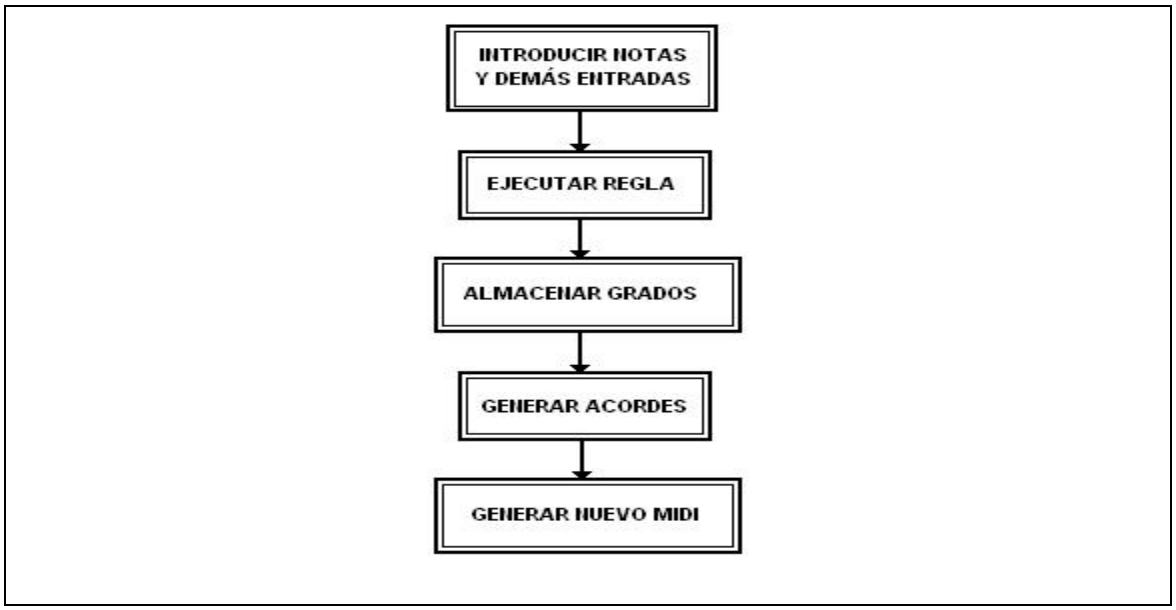


Figura 20: Diagrama de flujo B

Después de haber finalizado el análisis de la melodía inicial, se prepara el contexto para poder aplicar las reglas de armonía y obtener el acompañamiento. El contexto lo componen un vector de grados, correspondientes a las notas almacenadas en el vector que se ha generado en el proceso anterior, y las entradas o *facts* necesarios para desencadenar todo el funcionamiento de las reglas. Los grados se representan con números del 1 al 7 en decimal, y se obtienen mediante un método que será explicado en la implementación del sistema, llamado *getGrado*.

Una vez se ha generado el contexto, se llama a la regla, la cual devuelve la secuencia de grados, dentro de la tonalidad, que se corresponden con los acordes adecuados para el conjunto de notas que se le facilitó.

Los pasos que constituyen la ejecución de la regla se presentan en el siguiente diagrama.

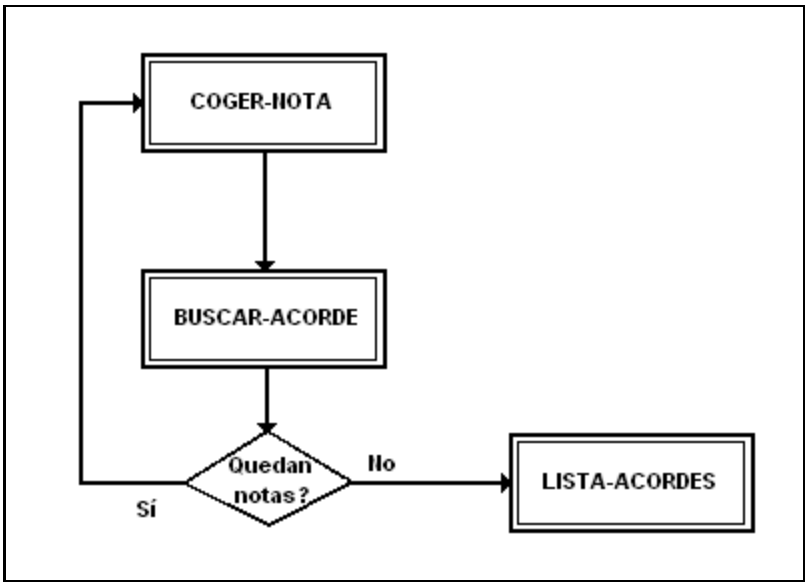


Figura 21: Diagrama de flujo de la regla

Como se puede observar en el anterior diagrama, el primer paso de la regla consiste en coger una nota de la serie generada por el proceso descrito en el diagrama A.

El siguiente paso, es obtener el grado del acorde dentro de la tonalidad de la pieza, correspondiente al grado de la nota en cuestión.

Este proceso se repite con todas las notas de la serie, hasta que no quede ninguna, en cuyo caso se procede a generar el objeto Java de la clase Acordes, en el cual se almacena la secuencia de grados que representan los acordes de acompañamiento.

Con los grados devueltos por la regla, y conociendo la tonalidad, se puede construir el acompañamiento. Cada acorde tendrá la duración que tenga el grupo con el que se corresponda.

Una vez obtenidos los acordes, se ejecuta el último paso, que consiste en generar una nueva voz y añadirla a la melodía original. Una vez hecho esto ya se puede concluir el proceso de composición terminando de generar el fichero MIDI de salida.

4. IMPLEMENTACIÓN DEL SISTEMA

4.1 Descripción del código

En este apartado se va a explicar con mayor detenimiento las partes del código que realizan las tareas presentes en los diagramas de flujo que se describieron en el apartado anterior.

Para ello se empezará por enumerar las clases que componen el sistema y los métodos que constituyen la parte funcional del compositor:

- Clases:
  - Compositor: es la clase principal, que contiene el *main*. En ella están implementados la mayor parte de los métodos que constituyen el compositor.
  - MIDI_Original: en ella se generan los ficheros MIDI originales que serán analizados. Tiene su propio *main*.
  - Acordes: esta clase sirve como intermediaria entre el programa Java y la parte de inteligencia artificial, es decir, la regla.
  - Regla: contiene la parte de inteligencia artificial, esto es, la base del conocimiento. No es una clase Java en sí misma, sino un fichero Jess. Sin embargo, puede ser comparable a una clase.

Los métodos se presentan en la siguiente tabla.

Método	Parámetros	Resultado	Clase
main	**String[]** args	void	Compositor
regla1	**Vector** notas, **String** tonalidad, **String** modoTonalidad, **String** fich_Midi	void	Compositor
regla1_ Paso1	**Vector** notas, **String** tonalidad, **String** modoTonalidad	Vector	Compositor
	Vector pieza, **String** tonalidad,		

regla1_Paso2	**String** modoTonalidad	Vector	Compositor
crearAcorde	**char** grado, **double** duracion, **String** modoTonalidad, **String** tonalidad	Note[]	Compositor
getNota	**char** grado, **String** tonalidad, **String** modoTonalidad	int	Compositor
getGrado	**String** tonalidad, **String** modoTonalidad, **int** tono_nota	int	Compositor
comprobarAcorde	**Note** nota, **String** acorde, **Note** primeraAcorde, **String** tonalidad, **String** modoTonalidad	boolean	Compositor
modoAcorde	**Note** nota, **String** modoTonalidad, **String** tonalidad	String	Compositor
escribeMidi	**Vector** duracion, **Vector** lista_acordes, **String** modoTonalidad, **String** tonalidad, **String** midi_ant	void	Compositor
leeMidi	**String** fich_midi	Vector	Compositor
main	**String[]** args	void	MIDI_Original

Tabla 6: Métodos del sistema

En la tabla anterior no se han incluido las definiciones de reglas del fichero Jess que componen la regla armónica puesto que no son métodos como tal, a pesar de que el concepto que representan en el fichero es semejante al de un método de una clase Java.

En este punto se va a explicar el funcionamiento del compositor a un nivel más bajo que con los diagramas de flujo del apartado anterior. Se van a describir más en detalle cada clase así como las entradas, salidas, funcionalidad e interacciones de sus métodos.

MIDI_Original

Esta clase es la que se utiliza para generar los ficheros MIDI que contienen la melodía que se quiere analizar y para la que se quiere construir un acompañamiento.

Está compuesta por un único método, el *main*. En él lo primero que se hace es crear un objeto *Score* que se utiliza para construir la pieza. A continuación se genera un *Array* de *Double* en el que se escribe la pieza en sí, especificando la nota y su duración.

Una vez escrita, se especifica el *tempo* de la misma, es decir, la velocidad a la que se reproducirá.

A continuación se vuelca el *Array* que contiene la melodía en un objeto *Phrase*, que será introducido a su vez en un objeto *Part*.

Por último, se guarda el *Part* en el *Score* y con ello queda escrito el fichero MIDI que será estudiado por el compositor.

Por tanto, se puede concluir que el fichero MIDI tiene siempre la siguiente estructura fija: consta del nivel superior formado por el *Score*, el cual contiene una parte (*Part*), y éste una frase (*Phrase*) donde se almacena la secuencia de notas que constituyen la pieza en sí.

Acordes

La clase *Acordes* está compuesta únicamente por una propiedad de la clase *String*, *listaAcordes*, y un constructor donde se le asigna un valor a ésta.

Esta clase actúa como el punto de unión entre la regla y el resto del programa. Permite recuperar el resultado generado por ésta para utilizarlo posteriormente.

Este enlace entre ambas partes se realiza mediante un objeto de la clase en cuestión, construido en uno de los últimos pasos de la regla. En Jess se pueden generar objetos Java de manera sencilla, y eso es lo que se hace. El resultado de la regla armónica se almacena en la propiedad del objeto *Acordes* antes mencionada, *listaAcordes*, en el momento de su creación, y se

devuelve como una cadena (*String*) de Java que contiene los acordes que constituyen el acompañamiento dentro de un objeto *Iterator*.

Regla

La regla no es una clase en sí misma, sino un fichero implementado mediante Jess. Está constituida por tres reglas definidas cuya interacción permite la construcción del resultado con los acordes finales: *coger-nota*, *buscar-acorde* y *lista-acordes*.

La primera de las definiciones, *coger-nota*, se encarga de recorrer una lista de notas, que son las que se obtuvieron mediante el bloque de control de regla explicado en apartados anteriores, y que se corresponden con las primeras notas de cada grupo (o frases melódicas), e ir cogiendo una a una dichas notas. Esta regla se ejecuta siempre en primer lugar para separar la nota que va a ser analizada del resto. El código Jess correspondiente se muestra a continuación.

```
(defrule coger-nota
"Va cogiendo nota a nota de la pieza para asignar los acordes
uno a uno"
?pieza-ant <- (pieza-nueva si)
?pieza-vieja <- (pieza ?prim $?resto)
?primera <- (primera-vez ?vez)
 =>
 (retract ?pieza-vieja ?primera ?pieza-ant)
 (assert (pieza $?resto))
 (assert (nota ?prim))

 (if (neq (length$ $?resto) 0) then
      (assert (buscar si))
      (if (eq ?vez 1) then
      (assert (acorde-anterior X))
      (assert (acorde-ant-anterior X))
  )
```

```
  (assert (primera-vez 0))
  (assert (pieza-nueva no))

        else
           (assert (buscar si))
           (assert (pieza-nueva no))
           (assert (imprime-acordes si))
  )
)
```

Para ejecutar esta regla es necesario haber hecho una serie de *asserts*, es decir, especificar unos hechos concretos de tal modo que cuando se hayan especificado todos se ejecute la regla. Estos *asserts* son los que en el código se nombran como *pieza* (almacena los grados de las notas a analizar. Es una secuencia de números del 1 al 7 como se mencionó en el apartado 3.11), *pieza-nueva* (se le da el valor *si* hasta que se haya obtenido la última nota de la lista, puesto que esta regla tiene que ejecutarse para obtener la nota que va a ser analizada en cada momento) y *primera-vez* (cuando se ejecuta la regla por primera vez hay que inicializar el valor del acorde anterior y del anterior al anterior de la primera nota, ya que aunque realmente no entren dentro de la secuencia final de acordes, se utilizan para decidir si se utiliza el acorde *correspondiente* o *reemplazante*. Una vez ejecutada la primera vez, se le da el valor *no*).

Aparte de coger nota a nota, comprueba si a la pieza que se le ha pasado le quedan notas por coger (valor almacenado en la longitud de la variable *$¿resto*). En caso de no quedar ninguna nota realiza los *asserts* necesarios (dando el valor *si* a *imprime-acordes*) para ejecutar el último paso, es decir, la regla *lista-acordes*. Si por el contrario, quedan una o más notas, se especificarán los *asserts* necesarios para ejecutar la regla *buscar-acorde*.

La definición de regla *buscar-acorde* es la que realmente constituye la regla armónica, ya que es la que aplica ésta sobre la nota en cuestión. Coge la nota que separó la definición *coger-nota* y elige un acorde adecuado para ella,

en forma de grado dentro de la tonalidad, tal y como se especificó en la tabla 4. La implementación correspondiente a esta parte de la regla es la siguiente.

```
(defrule buscar-acorde
"Asigna un acorde a cada nota de la pieza"
     (buscar si)
     ?acordes-viejos <- (acordes $?acor)
     ?acorde-ant <- (acorde-anterior ?acor-ant)
     ?acorde-ant-ant <- (acorde-ant-anterior ?acor-ant-ant)
     ?nota-ant <- (nota ?nota)
     =>
     (retract ?acordes-viejos ?acorde-ant ?acorde-ant-ant ?nota-ant)
     (assert (pieza-nueva si));----------------------------------para
activar "coger-nota"

     (if(and(eq ?nota 1)(or(and(eq ?acor-ant 0)(eq ?acor-ant-ant 1))
     (eq ?acor-ant 1))) then
           (assert (acordes $?acor 4))
           (assert (acorde-anterior 4))
           (assert (acorde-ant-anterior ?acor-ant))

        else
           (if (eq ?nota 1) then
              (assert (acordes $?acor 1))
              (assert (acorde-anterior 1))
              (assert (acorde-ant-anterior ?acor-ant))
           );--------------------------------------else 1
```

Solamente se muestra el código implementado para el caso de decisión en que el grado de la nota analizada es “1”. El resto de casos es igual salvo que se cambia el grado de la nota y el grado del acorde correspondiente.

El proceso de selección del acorde *correspondiente* o *reemplazante* en función del grado de la nota analizada, se describe a continuación.

En primer lugar, se toma la nota obtenida mediante la regla anterior. Esta nota no es realmente una de las notas denominadas *primera nota de grupo*, que se obtuvieron en pasos anteriores, sino que es el grado que le

corresponde a una de ellas en la tonalidad en la que se esté. A continuación se realiza la siguiente comprobación:

(if (and (eq ?nota 1) (or (and (eq ?acor-ant 0) (eq ?acor-ant-ant 1)) (eq ?acor-ant 1)))

Si se traduce esa sentencia condicional de Jess a Java, queda lo siguiente:

if((nota==1) && (((acor-ant==0)&&(acor-ant-ant==1)) | | (acor-ant==1)))

Esta condición comprueba si el grado de la nota es 1, y si además el acorde anterior que se decidió es grado 1 o en caso de que éste fuese un silencio, que el anterior al anterior fuese grado 1. Esto es necesario para no utilizar dos veces seguidas el mismo acorde, por razones melódicas.

Anteriormente se comentó un poco la inicialización de las variables *acor-ant* y *acor-ant-ant*, pero no se especificó por qué se les da el valor X al comienzo. Esto se debe a que, como se ha podido ver, ambas variables son necesarias para seleccionar el acorde que se va a utilizar en cada momento, y en el caso de la primera nota, no se habrá especificado todavía ningún valor para el acorde anterior ni el anterior al anterior, o sólo para el anterior al anterior en el caso de la segunda nota. De ahí que se les dé un valor arbitrario inicial.

Una vez ha elegido el acorde, realiza los *asserts* necesarios para que se vuelva a ejecutar el primer paso, *coger-nota*.

Cuando se han terminado de analizar todas las notas, se ejecuta la definición *lista-acordes*. Ésta, lo que hace es construir el objeto de la clase *Acordes*, para que la cadena con los acordes generados pueda ser accedida desde la parte Java del compositor. Su código Jess es el que sigue.

```
 (defrule lista-acordes
 "Imprime los acordes y crea el objeto JAVA con la lista de
acordes"

    ?imprime-ant <- (imprime-acordes si)
    (acordes $?acord)
    (pieza-inicial $?notas)
    =>
    (if (eq (length$ $?notas) (length$ $?acord)) then
       (retract ?imprime-ant)
       (bind ?string (implode$ $?acord))
       (add (new Acordes ?string))
    )
)
```

Una vez se ejecuta esta definición, finaliza la ejecución de la regla y se le da paso, de nuevo, al *bloque de control de regla*, para que procese el resultado.

Compositor:

Esta clase es la que contiene toda la lógica del *bloque de control*, del *bloque integrador*, del *escritor archivos MIDI* y del *lector*.

A continuación se muestra un diagrama de flujo en el que se presentan todas las interacciones existentes entre los métodos de la clase compositor, así como las diferentes variables intermedias que se intercambian.

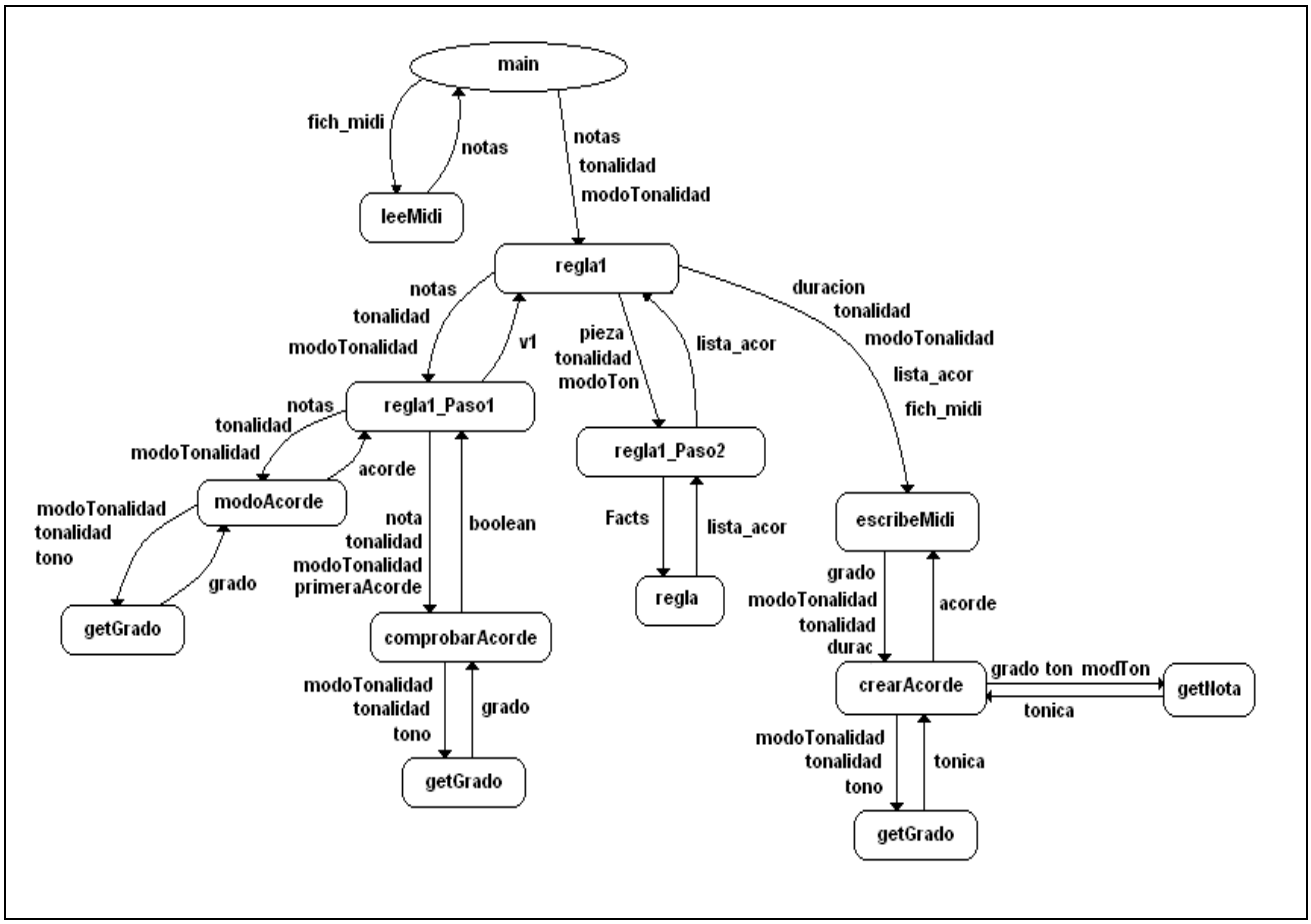


Figura 22: Diagrama de flujo de los métodos del sistema

En el diagrama anterior se han dibujado los métodos por orden de ejecución. Este orden va de arriba hacia abajo y de izquierda a derecha. Por ejemplo, a partir de *regla1* se ejecutan, por este orden, los siguientes métodos: en primer lugar se ejecuta *regla1_Paso1*, por estar más arriba y a la izquierda, y todos los métodos que descienden desde él por estar más a la izquierda, después le toca el turno a *regla1_Paso2* y lo que descienda de él, y por último *escribeMIDI* y lo que de él descienda.

El método inicial del que parte todo es el *main*. Este método se encarga de recoger los parámetros de entrada al programa (tonalidad, modo de la misma (mayor o menor) y el fichero MIDI que se desea analizar para desarrollar su acompañamiento), de leer el fichero que se ha seleccionado y de elegir qué regla aplicar en caso de que hubiese varias.

En este proyecto sólo se ha especificado una regla, por lo que se ha omitido la entrada que se correspondería con el tipo de acompañamiento que se desease generar y por tanto que permitiría seleccionar qué regla aplicar.

La tonalidad se ha definido como un parámetro de entrada, puesto que no es una tarea de este compositor. Este trabajo de obtención de una tonalidad no es trivial y bien podría ser un proyecto previsto como un módulo más de este compositor. Lo mismo ocurre con el modo de la tonalidad.

El fichero MIDI se especifica por línea de comandos, y viene ya generado por la clase que se explicó con anterioridad, *MIDI_Original*. El primer paso llevado a cabo por el *main* es leer este fichero. Para ello hace una llamada al método *leeMidi*.

Este método recibe como parámetro de entrada el fichero MIDI y devuelve como parámetro de salida un vector de notas, que son las que componen la melodía. La lectura del MIDI se realiza siempre de la misma manera, en sentido inverso a como se generó.

El primer paso es generar un objeto *Score* para poder volcar el fichero y así extraer sus componentes. Después se obtiene la parte (objeto *Part*) y de ella la frase (objeto *Phrase*). Finalmente se recuperan las notas que se encuentran contenidas en el objeto *Phrase*. Por esto es importante guardar siempre la misma estructura en los ficheros MIDI que se generan, para poder hacer una lectura genérica de todos ellos sin tener que modificarla para cada uno en concreto.

El vector devuelto se almacena en otro vector denominado *notas*, que se pasará como parámetro del siguiente método a ejecutar.

La siguiente llamada que hace el *main*, es al método *regla1*. A pesar de su nombre, este método no es la regla en sí, pero contiene la lógica encargada de preparar su ejecución así como de llamarla. A partir de este punto se

desencadena toda la funcionalidad del compositor hasta la escritura del fichero final.

Los parámetros que recibe como entradas son, el vector *notas* que se obtuvo con la llamada a *leeMidi*, la tonalidad, el modo de ésta y el fichero MIDI original, puesto que aunque ya haya sido leído, es necesario en pasos posteriores para la escritura del fichero final.

El primer paso consiste en llamar a *regla1_Paso1*, cuyos argumentos de entrada son los mismos que los de *regla1* salvo el fichero MIDI original.

En este método se lleva a cabo todo el proceso explicado en el diagrama de flujo que se denominó *Bloque A*, en el apartado 3.11. Puesto que su funcionamiento ya se explicó anteriormente, se va a especificar directamente cuáles son las llamadas que realiza a otros métodos y cómo funcionan estos métodos.

La primera llamada que realiza es a *modoAcorde*. Este método recibe tres argumentos, una nota, la tonalidad y el modo de la misma. La nota se corresponde con la que será una *primera nota de grupo*.

En primer lugar, *modoAcorde* coge el tono de la misma, para llamar al método *getGrado,* que lo recibe como entrada para obtener el grado de dicha nota, y una vez lo ha recuperado comprueba cuál es el modo del acorde de la nota que se pasó como parámetro.

En una determinada tonalidad, los acordes asociados a cada nota de la misma pueden ser mayores o menores, pero sólo uno de los dos modos. Éste va ligado al grado con que se corresponde la nota dentro de la tonalidad y al modo de la propia tonalidad.

Por ello, *modoAcorde* necesita el grado de la nota recibida, y en función de éste, devuelve el modo tras una sucesión de *if-else*.

El método *getGrado*, como ya se ha mencionado, se encarga de obtener el grado de una nota a partir del tono de la misma, la tonalidad y el modo de ésta, que son los tres parámetros de entrada. En primer lugar fija cuál es la nota tónica en función de la tonalidad y su modo. Y a continuación, calcula el resto del cociente formado por la resta de los enteros correspondientes al tono de la nota en cuestión y de la tónica, y el número 12. Esto se debe a que, en JMusic, los tonos tienen asociado un entero, y la distancia de una nota a su octava es 12 y cada semitono es una unidad más. De esta manera, si se resta cualquier tono dentro de una octava a la tónica en esa octava, se va a obtener un número menor que 12 que dividido por él dará como resto la diferencia. En caso de que la nota esté en alguna escala por encima de la tónica, la diferencia será un número superior a 12, pero será justamente 12 más la diferencia que habría entre el tono correspondiente a esa nota dentro de la octava de la tónica y la tónica. Así el resto obtenido en el cociente entre éste número y 12 es un número entre 0 y 12 al igual que en el caso anterior.

Con este resultado, se puede saber con qué grado se corresponde la nota. En la siguiente tabla se muestran estas correspondencias:

Resto	0	2	4	5	7	9	11
Grado	I	II	III	IV	V	VI	VII

Tabla 7: Par grado-resto

Por ejemplo, si la tonalidad con la que se trabaja es Do mayor, y queremos saber qué grado le corresponde al Re de la escala siguiente a la central:

$$\left.\begin{array}{l} C4 \rightarrow 60 \\ D5 \rightarrow 74 \end{array}\right\} |60-74| = 14 \Rightarrow 14\%12 = 2 \Rightarrow II \text{ Grado}$$

Para una persona, es evidente que cualquier Re, esté en la escala que esté, es el grado II, sin embargo, el ordenador tiene que calcular siempre cuál es el grado de una nota, puesto que no trabaja con nombres sino con números.

Una vez se ha obtenido el grado, se devuelve para que *modoAcorde* pueda decidir qué modo le corresponde al acorde asociado con la nota de entrada.

Después de realizar estas operaciones, el siguiente método al que llama *regla1_Paso1* es a *comprobarAcorde*. La función de este método es verificar si las notas siguientes a la *primera nota de grupo* pertenecen o no a su acorde, es decir, si son tónica, tercera o quinta para establecer el grupo.

Para ello, se le pasan como argumentos la nota que está siendo analizada, la *primera nota de grupo*, el modo del acorde de la *primera nota de grupo* calculado previamente, la tonalidad y el modo de la tonalidad.

En primer lugar se calcula para la nota y la *primera nota de grupo* el resto obtenido del cociente de sus tonos respectivos con 12.

Tras este cálculo, se obtienen la distancia existente entre la tercera y la primera, y la quinta y la primera del acorde, en función del modo de éste. Estas distancias son constantes y dependen del modo del acorde, de tal manera que la distancia para una tercera mayor (2 tonos) es cuatro, ya que cada semitono equivale a una unidad de separación en JMusic, y para una tercera menor (1 tono y medio) es tres.

En la siguiente tabla se presenta una tabla con las distancias correspondientes a la tercera y la quinta.

Acorde	Tonalidad	Grado nota	Distancia tercera	Distancia quinta
M	M	~	4	7
M	m	~	4	7
m	M	VII	3	6
m	m	II	3	6

m	M	!=VII	3	7
m	m	!=II	3	7

Tabla 8: Distancia de tercera y quinta con la primera del acorde

A continuación se suman respectivamente al resto de la *primera nota de grupo* con 12. De esta manera se obtienen los enteros exactos correspondientes a los tonos de las notas del acorde.

Finalmente se comprueba si el resto del cociente formado por la nota que se está analizando y 12, calculado en el primer punto, se corresponde con alguno de los enteros calculados en el segundo paso. Si no es así, se devuelve un *buleano* con valor *false*, y en caso contrario con valor *true*.

Los cálculos realizados en el paso dos, se hacen para poder llevar a cabo la comprobación sin errores, puesto que la tercera y la quinta de un acorde tienen una separación con respecto a la primera constante y dependiente del modo del acorde, y en caso de que tuviésemos la primera en una octava, y la siguiente nota fuese la tercera pero una octava por encima, la comparación no sería correcta.

Por último, *regla1_Paso1* almacena en la primera posición del vector devuelto, el vector en el que se encuentra la secuencia de notas que constituyen las *primeras notas de grupo*, y en la segunda posición el vector que contiene las duraciones de cada grupo.

Una vez se han realizado estas operaciones, *regla1* recupera del vector devuelto por *regla1_Paso1* la secuencia de notas, almacenada en el vector *pieza*, y las duraciones, almacenadas en el vector *duracion*, y llama al segundo paso de la regla, *regla1_Paso2*.

Este método recibe como argumentos la secuencia de notas, es decir, el vector *pieza*, la tonalidad y el modo de ésta. En primer lugar, genera los *facts* y

realiza las operaciones necesarias para llamar a la regla y desencadenar todo el proceso de análisis de la secuencia de notas.

Tras llamar a la regla, coge el resultado devuelto. Este resultado es un objeto de la clase *Acordes*, que está contenido en la primera y única posición de un *Iterator*.

De este objeto *Acordes* se obtiene su atributo, la cadena que contiene la lista de los acordes, representados por el grado de la primera nota de los mismos. La lista tiene un formato en el que cada grado viene representado por un número romano de I a VII, y cada uno está separado del siguiente por un espacio. A continuación se separa cada grado en una posición de un *Vector* y se devuelve éste.

Por último, *regla1* llama a *escribeMidi*. Este método se encarga de realizar las últimas operaciones para poder escribir el fichero de salida.

Los parámetros de entrada que recibe son *duracion*, que es el vector que contiene las duraciones de los acordes, *lista_acordes*, vector con la secuencia de grados, la tonalidad, el modo de la tonalidad y el fichero MIDI original.

En primer lugar, crea un objeto *Score*, para volcar el fichero MIDI original en él, y posteriormente añadirle el acompañamiento.

A continuación va recorriendo los vectores *duracion* y *lista_acordes*, y va extrayendo una a una sus componentes y llama al método *crearAcorde*. Este método devuelve un array de tres notas, que son las que componen el acorde del grado correspondiente a la posición actual de *lista_acordes*.

Una vez ha obtenido todos los acordes en su orden secuencial y con su duración, los introduce en una parte nueva y añade ésta al *Score*.

Finalmente vuelca en el nuevo fichero el *Score*, obteniendo así el fichero de salida.

El método *crearAcorde*, llamado por *escribeMidi*, se encarga de generar los acordes a partir de los grados que se encuentran en *lista_acordes*. Para ello, obtiene el tono de la nota correspondiente al grado que se le pasa como parámetro, mediante el método *getNota*, que se explicará a continuación. Esta nota que se obtiene, es la tónica del acorde.

Tras obtener la tónica, comprueba cuál es su grado y cuál es el modo de la tonalidad, y en base a ellos construye el acorde con la tónica y añadiendo a ésta la tercera y la quinta. Es importante conocer el grado y el modo de la tonalidad, porque en función de ellos la distancia entre la tónica y la tercera o la tónica y la quinta son diferentes. De esta manera, para generar la tercera y la quinta se le sumará a la tónica la distancia adecuada y añadiremos la nota al acorde.

El método *getNota* se encarga de realizar la función inversa a *getGrado*, es decir, obtiene el tono de una nota a partir del grado que le corresponde dentro de la tonalidad.

En primer lugar, obtiene la tónica en función de la tonalidad. La tónica que se utiliza es la de la escala número 3 de JMusic, puesto que es la que se corresponde con la escala que contiene el Fa abrazado por la Clave de Fa, y su tono es suficientemente grave para realizar un acompañamiento que se diferencie de la melodía.

Una vez ha obtenido la tónica, para obtener el tono de la nota lo que hace es comprobar el grado de ésta, y en base a él le suma a la tónica la distancia adecuada para alcanzar ese grado.

Finalmente, cuando ya ha calculado el tono de la nota, lo devuelve para que lo pueda utilizar *crearAcorde*.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones

En este proyecto se ha realizado un compositor musical que genera un acompañamiento constituido por una secuencia de acordes, los cuales se han construido a partir del análisis armónico de una melodía de entrada.

El sistema desarrollado ha alcanzado los objetivos previstos en su inicio. Por un lado, se ha creado la plataforma lógica necesaria para seleccionar qué conjunto de reglas aplicar sin modificar la base del conocimiento. Simplemente se ha de especificar qué tipo de composición se desea, y el propio sistema se encarga de acceder a la parte de la base de conocimiento correspondiente.

Y por otro lado, se ha generado un conjunto de las mismas que demuestra la capacidad inteligente del compositor. Esta capacidad inteligente aplica un determinado conocimiento musical, fijado por el mencionado conjunto de reglas, para llevar a cabo la composición de una línea de acompañamiento a partir de una melodía dada.

El compositor se divide de manera teórica en una serie de bloques, los cuales tendrán, cada uno, una función dentro del sistema y que se ejecutarán de manera secuencial. Cada bloque utilizará como entrada la salida del anterior, y generará una salida que el siguiente bloque recogerá para desarrollar su función dentro del proceso.

El primero de los bloques es el lector de los archivos de audio. Este bloque analiza la melodía de entrada, obteniendo la serie de notas que la componen. El siguiente bloque, es el decisor de reglas, que se encarga de seleccionar la regla a aplicar en base a la elección del usuario. Éste, al realizar la selección de la regla, llama realmente al bloque controlador, puesto que primero se necesita un procesamiento de la melodía para poder aplicar la regla musical. Una vez se han preparado los datos necesarios, se ejecuta el bloque

correspondiente a la regla. Ésta, devuelve un resultado con los grados de los acordes que se constituyen el acompañamiento, que es recogido por el bloque de control y entregado por éste al generador del archivo MIDI final.

Otra forma de analizar el sistema compositor es de manera física, es decir, atendiendo al código que realmente se ha implementado. De esta manera se tiene que los distintos bloques teóricos en que se ha dividido el compositor, se corresponden con diferentes porciones del programa.

Las herramientas utilizadas para realizar todo el proyecto, Java, Jess y JMusic, han sido verdaderamente útiles y se han ajustado bastante bien a los requerimientos de este trabajo, permitiendo solventar todos los problemas que se han planteado.

La herramienta más utilizada ha sido JMusic. Su empleo es bastante intuitivo y sencillo, y además consta de una documentación muy completa en la que se explica paso por paso las partes, clases y demás utilidades de la librería. La mayor parte del proyecto se ha implementado con JMusic, ya que con él se ha realizado la lectura y escritura de ficheros MIDI y se ha procesado la información de dicho fichero para preparar la secuencia de notas que serán analizadas por la regla.

El lenguaje de inteligencia artificial, Jess, es también bastante intuitivo aunque no fácil de utilizar y dispone de una documentación muy completa. Jess está dotado de los instrumentos necesarios para realizar sistemas de reglas de todo tipo, tanto simples como complejos, pero su semejanza con los lenguajes *script* hace que, por pequeño que sea el sistema a implementar, éste se convierta en un complicado entramado de definiciones de reglas y/o funciones. En el caso de este proyecto, la regla que se ha implementado no tiene la complejidad necesaria como para aprovechar al máximo esta herramienta. Probablemente, en otro tipo de sistemas de mayor envergadura, como puedan ser aplicaciones industriales, sí se puede sacar un mayor rendimiento a este lenguaje de inteligencia artificial.

En mi opinión la composición realizada no alcanza una complejidad musical excesiva. Sin embargo, su implementación no es ni mucho menos fácil, puesto que la música es algo intrínseco al ser humano, y aunque esté reglada y se hayan creado múltiples teorías armónicas que puedan ser plasmadas mediante algoritmos, no deja de tener un alto contenido emocional y sensitivo, los cuales no están presentes en una máquina.

Una de las partes más difíciles que han surgido a la hora de realizar el proyecto, ha sido decidir cuál es la longitud de las frases musicales. Una persona, aunque no tenga nociones musicales de ninguna clase, puede deducir más o menos, mediante el simple hecho de escuchar una pieza, dónde empieza y termina una frase, es decir, dónde comenzaría un acorde y dónde terminaría éste para dar paso a otro o incluso a ninguno. La longitud de la frase puede variar, no siempre será la misma, por mucho que existan reglas que dictan la división del compás, el tempo y demás. Por ello, una máquina, la cual no está dotada de sentidos ni emociones, solamente puede decidir la conclusión de una frase mediante la aplicación de algún algoritmo que no podrá reflejar ninguna de las dos cualidades mencionadas, como mucho podrá aproximarse un poco.

La solución a este problema, aportada por este proyecto, se puede dividir en dos partes. Una primera parte en la que se le permite al usuario decidir cuál es la longitud máxima de la frase que desea, lo cual sí puede determinarse en cierto modo basándose en la división y el compás utilizado en la melodía. Así en un cuatro por cuatro en el que se utilicen fundamentalmente figuras de corta duración, como la corchea, se puede establecer como longitud máxima de las frases una negra o una blanca, en base a la sensación de dinamismo que se le pretenda dar a la pieza. Por ello, no es la máquina quien tiene todo el poder de composición. Parte de este proceso se le cede al usuario.

La otra parte de la solución es una tarea del ordenador. Una vez se ha establecido la longitud máxima de la frase, es el ordenador quien decide si se llega hasta esa longitud o si no por el contrario se corta antes. Para ello se basa en pruebas realizadas frente al piano, en las que se iban tocando secuencias de notas con la mano derecha y mantenía un acorde pulsado con la izquierda, coherente musicalmente con la primera de aquellas, hasta que se detectaba una discordancia. Se concluyó que el acorde correspondiente a una nota, el cual no tiene porqué ser el constituido por ella, su tercera y su quinta, se podía mantener pulsado aproximadamente durante dos notas más, si éstas no pertenecen al acorde de la primera. Ésta es la idea en la que se basa el ordenador para decidir la longitud de la frase. Sin embargo, esto es muy limitado, puesto que puede haber muchos más criterios que una persona pueda tomar en base a lo que oye, mientras que el ordenador se mantiene en uno fijo.

Por ello, hoy por hoy, a no ser que se consiga reconstruir el cerebro humano con máquinas, su forma de pensar, sentir, etc., no se va a poder crear un compositor musical real, capaz de dar lugar a piezas melódicamente comparables con cualquiera que haya creado uno de los grandes músicos de la historia. Quizá se consiga obtener buenos resultados melódicos, pero siempre basado en reglas y pautas, de manera que se conseguirá una música estructuralizada, que tenderá a ser siempre parecida.

Esto no implica que la música realizada por ordenador no tenga utilidad. Este tipo de composiciones computerizadas se puede emplear en el sector de la música comercial, en la cual se siguen siempre unas pautas rítmicas y melódicas para obtener unos resultados que agraden fácilmente al oído. También podrían implementarse programas que hiciesen arreglos en piezas musicales, puesto que para la corrección musical sí existen reglas de armonía fijas, que pueden ser aplicadas de una forma más rápida y eficiente por una máquina.

La otra gran dificultad del proyecto fue el planteamiento en general de cómo llevar a cabo el compositor, qué bloques serían necesarios y qué clases y métodos son los que habría que implementar.

Finalmente se puede concluir que los resultados de este compositor son satisfactorios, puesto que se ha conseguido construir el compositor como tal, es decir, un sistema que genere un acompañamiento a partir de una secuencia melódica dada, y también se ha generado la estructura básica que permite seleccionar las reglas a utilizar sin necesidad de modificar el módulo principal.

5.2 Trabajos futuros

Las posibles ampliaciones que este proyecto permite son enormes. Por ejemplo, se podría generar un módulo adicional que obtuviese la tonalidad y su modo a partir de la melodía, sin necesidad de introducirla como argumento.

Otra posibilidad es la creación de nuevas reglas de composición que se añadan a la que ya ha sido creada. Es decir, aumentar la base del conocimiento del sistema. Incluso podrían ir orientadas a estilos musicales determinados, como el Barroco, la música coral, etc., o imitando la obra de autores específicos como Beethoven, Mozart, etc. También se pueden añadir nuevos criterios para la selección de la longitud de la frase en la regla implementada en este proyecto.

Se podría dar la opción al usuario de que el compositor generase una melodía principal, sin ser el acompañamiento de una ya dada. Esto supondría la creación de un sistema muy complejo de reglas armónicas, y como se ha dicho antes, no sería una composición libre (fundamental para dar lugar a una composición humana), sino estructuralizada y siempre basada en pautas.

Otra ampliación más podría se la introducción de más voces generadas con diversos instrumentos, para dar la sensación de un acompañamiento orquestal.

Desde el punto de vista técnico, se podrían estudiar otras tecnologías y motores de sistemas expertos, como Jboss Rules (Drools), para comparar sus capacidades expresivas.

BIBLIOGRAFÍA

ALONSO, D. **"Sistemas expertos"**. 2001
http://es.geocities.com/denisalonso2001/SISTEMAS_EXPERTOS.htm

DION, J. A. **"Automated Music Composition: an Expert Systems Aproach"**. 2006
http://www.rivier.edu/journal/ROAJ-2006-Spring/J43-DION.pdf

DOBRIAN, C. **"Music and Artificial Intelligence"**. 1993
http://music.arts.uci.edu/dobrian/CD.music.ai.htm

GROUT, D. J. Y PALISCA, C. V. "Historia de la música occidental, 1". 1995

GRUPO ALIANZA EMPRESARIAL (GAE). 2009
http://www.grupoalianzaempresarial.com/inteligenciaartificial.htm

JESS. 2009
http://www.jessrules.com/

JORDÁ, S. **"Música e Inteligencia Artificial"**. 1990
http://www.iua.upf.es/~sergi/musicia.htm#LA%20INTELIGENCIA%20MUSICAL%20ARTIFICIAL

Mailxmail.com. **"Historia de la música"**. 2002
http://www.mailxmail.com/curso-historia-musica

MENKEN, M. **"Jess Tutorial"**. 2002
http://www.cs.vu.nl/~ksprac/export/jess-tutorial.pdf

MIRANDA, E. R. **"Basic Concepts of Digital Music and Computational models of music"**. 2003
http://x2.i-dat.org/~csem/UNESCO/

RASO DEL MOLINO, J. **"La armonía"**. 2000
http://www.filomusica.com/filo1/jr.html

REYNOSO, R. E. **"Introducción a la armonía tradicional"**.
http://akamai.www.berkleemusic.com/assets/display/8214291/8214290.pdf

SAMPER, J. J. **"Introducción a los sistemas expertos"**. 2009
http://www.redcientifica.com/doc/doc199908210001.html

SORENSEN, A. y BROWN, A. **"JMusic. Music composition in Java"**. 1998
http://jmusic.ci.qut.edu.au/

TARKO, V. **"How Music Influences Brain Development"**. 2006
http://news.softpedia.com/news/How-Music-Influences-Brain-Development-36063.shtml

Wikipedia. 2008
http://es.wikipedia.org/wiki/Sistema_experto